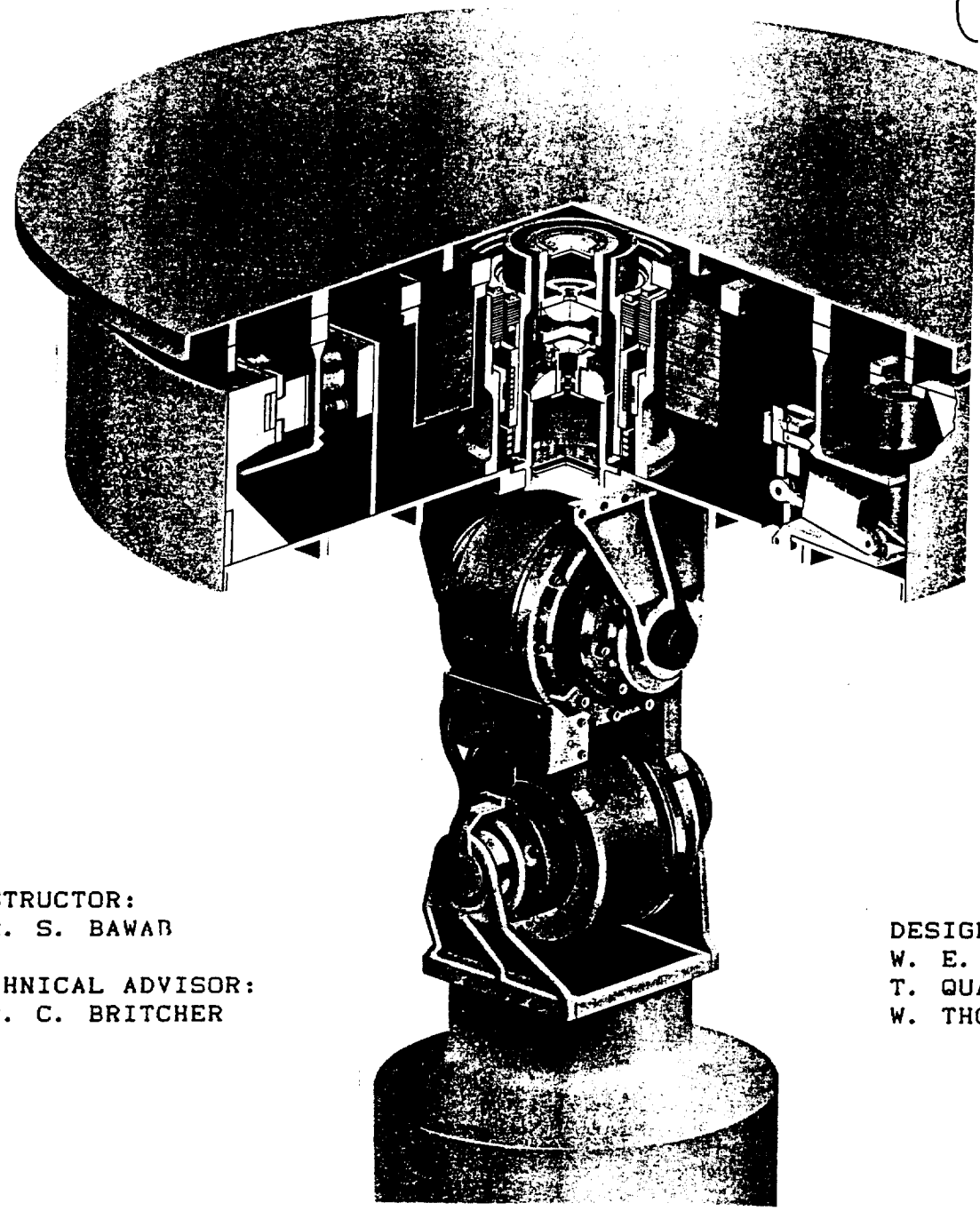
NASw-4435

ANNULAR SUSPENSION AND POINTING SYSTEM (ASPS) MAGNETIC ROTARY JOINT



INSTRUCTOR:
DR. S. BAWAB

TECHNICAL ADVISOR:
DR. C. BRITCHER

DESIGN TEAM MEMBERS:
W. E. SMITH
T. QUACH
W. THOMAS

SENIOR DESIGN PROJECT
OLD DOMINION UNIVERSITY
FALL 1993

OVERVIEW OF TEAM PARTICIPATION:

The magnetic rotary joint was initially divided into three lead functions. W.E. Smith served as manager, W. Thomas as lead hardware engineer and T. Quach as lead software engineer. This proved to be an effective and evenly distributed breakdown. The functions and involvement of each team member is as follows;

T. Quach - Software Engineer

* Developed C programs by learning the C language, installing required software, and writing appropriate code.
* Assisted in development of hardware configuration to ensure proper integration with software.
* Wrote and presented one status report.
* Developed mathematical model using MATLAB.

W. Thomas - Hardware Engineer

* Developed hardware configuration by reviewing system schematics, tracing signal data points, and wiring the control panel for digital control.
* Assisted in development of software configuration to ensure proper integration with hardware.
* Wrote and presented one status report.
* Developed mathematical model.

W.E. Smith - Manager

* Assisted and directed development of hardware and software elements.

* Met with technical advisor weekly.
* Met with managers bi-monthly.
* Held weekly meetings with team members.
* Wrote meeting minutes.
* Wrote and presented two status reports.
* Developed mathematical model.

A few rough edges in the report (due to running short of time) but very solid material and clearly a great deal of time and effort expended on this project.

A very good result from a quite challenging piece of work.

I concur with (A)

Colin Britcher.

TABLE OF CONTENTS

List Of Figures And Tables

I. ABSTRACT

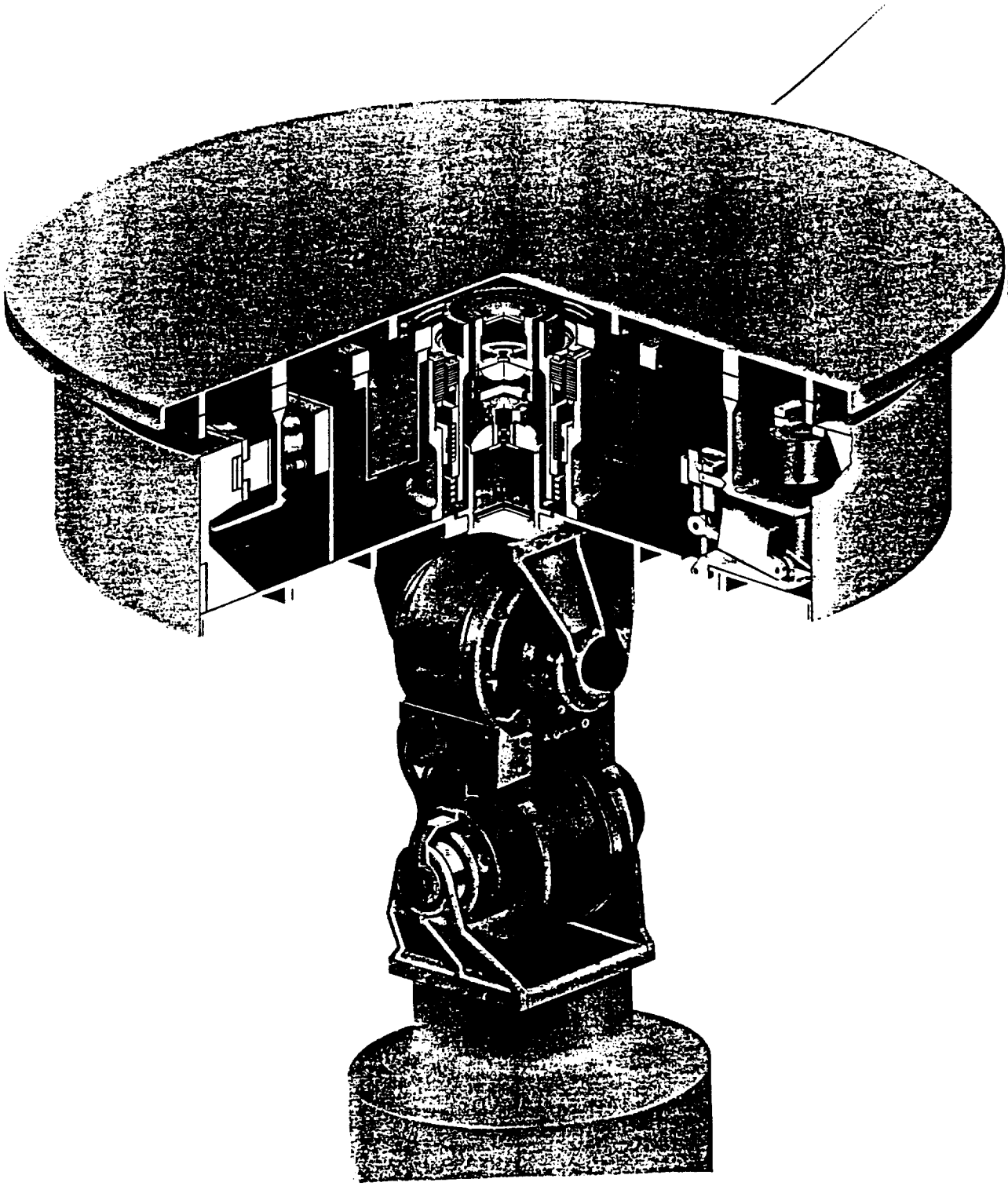
The Annular Suspension and Pointing System (ASPS) is a prototype of flight hardware for a high-accuracy space payload pointing mount. The long term project objective is to perform modifications and implement improvements to the existing ASPS in hopes of recommission. Also, new applications will be investigated for this technology. This report will focus on the first aspect of this overall goal, to establish operation of a single bearing station. Presented is an overview of the system history and bearing operation followed by the processes, results, and status of the single bearing study.

II. BACKGROUND

In mid 1976, the NASA Langley Research Center awarded the Sperry Corporation's Flight Systems Division a contract to develop an auxiliary pointing system. This system was to be capable of subarcsecond performance maintained in the carrier vehicle disturbance environment which consists of both vibrational and transient disturbances. The result of this contract is the annular suspension and pointing system. This system (figure 1) includes a magnetically levitated isolation and vernier pointing system attached to a modular gimbal.

The ASPS was delivered to NASA in 1983; however, it has never been operated by NASA. Due to shifting NASA priorities and difficulties in the space shuttle program, the ASPS program was terminated shortly after delivery. The system had been stored at NASA until it was loaned to ODU in the fall of 1992. The hardware is now situated in the digital/controls laboratory.

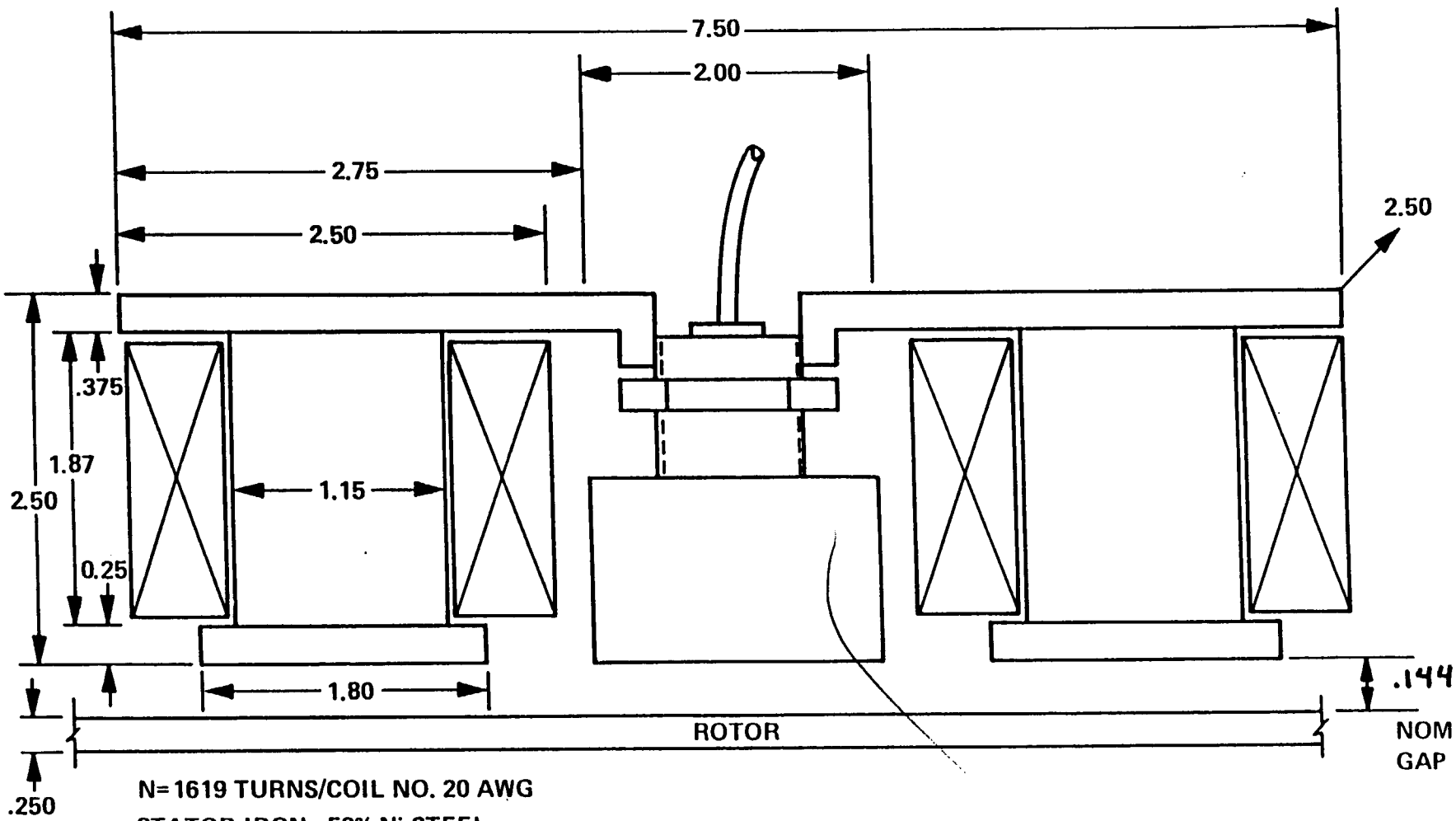
Figure 1. ASPS Magnetic Rotary Joint



1. DESCRIPTION AND OPERATION OF THE MAGNETIC BEARINGS

The magnetic bearing portion of the ASPS consists of a set of stationary electromagnets (stator) and a 50% nickel 50% steel annular ring (rotor). This configuration allows the ring and thus the payload, to be suspended in a magnetic field and to provide precise positioning (figure 2). The ring position is maintained dynamically using sensors to provide a continuous feedback loop to the electromagnetic poles. Three individual systems are integrated in this application to provide a six degree of freedom pointing mechanism. Three axial bearings are employed to control vertical positioning, two radial bearings control motion in the ring plane, and a roll motor provides tangential motion (figure 3). The advantages of such a system are numerous. The most evident being the lack of contact between bearing components and thus the elimination of wear. This results in extended life for the bearing components and less maintenance on the system. Additionally, this technology allows disturbances to the system to be isolated. These attributes make magnetic bearings ideal for space applications.

AXIAL MBA WITH SENSOR (HALF STATION)



N= 1619 TURNS/COIL NO. 20 AWG

STATOR IRON: 50% Ni-STEEL

ROTOR IRON: LOW CARBON STEEL

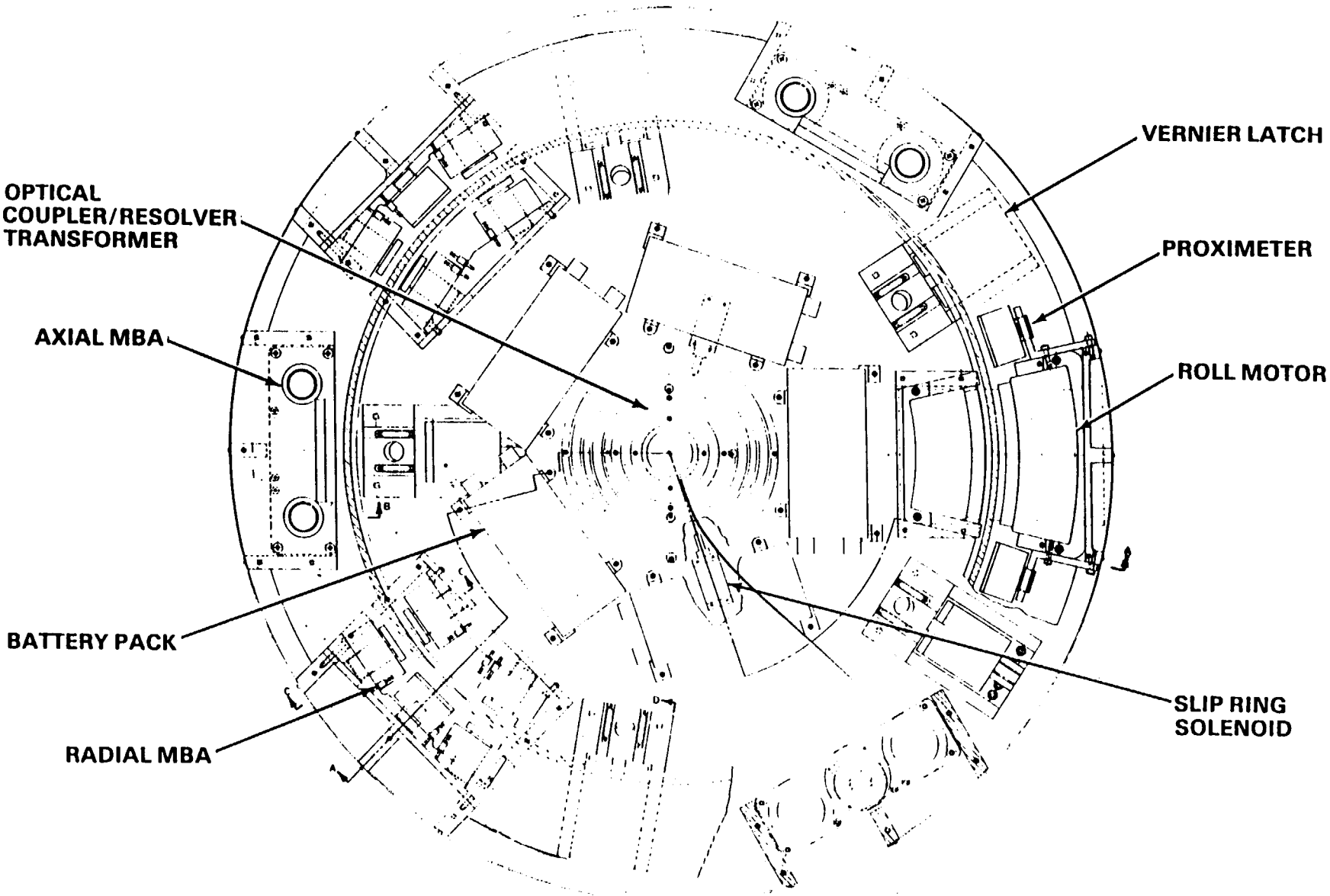
Go= 0.300

G= Go±0.220

Figure 2. Axial MBA

Figure 3.

VERNIER LAYOUT



2. MAIN PROJECT OBJECTIVE

Old Dominion University, through this project, desires to update the ASPS technology and bring the system back to full operating condition. In addition, the hardware will be used to investigate new applications for this type of magnetic suspension system.

3. PREVIOUS DESIGN MODIFICATIONS/ANALYSIS BY ODU RESEARCHERS

Some analysis of the existing ASPS circuitry was performed during the spring semester. As a result, the function of certain elements were identified. In addition, it was determined that a relatively simple modification to the gap distance would raise the vertical force capability to a satisfactory level for the 1g environment. Consequently, the parts for this modification have been fabricated.

III. SENIOR PROJECT OBJECTIVE AND APPROACH - FALL 1993

The goal of this design team is to make a single ASPS bearing station operational using digital indication and control.

To meet this objective, the project was initially divided into two areas of development (figure 4). The software area focuses on learning the C programming language, identifying the required program modifications, and writing new code. In parallel to this, the hardware aspect focuses on learning the existing system circuitry, testing elements for proper operation, and identifying the items required to operate a single station. In addition to these areas, development of the feedback control logic focuses on Proportional-Derivative (PD) type control and identifying the elements required to implement such logic.

These elements will be integrated (figure 4a) such that the C programming, containing the proper control logic, commands the electromagnet output through the electrical hardware. The computer then receives a feedback signal from the sensor which is fed back into the programming to readjust the electromagnet output and thus the annular ring position.

Figure 4. ASPS Work Control Network

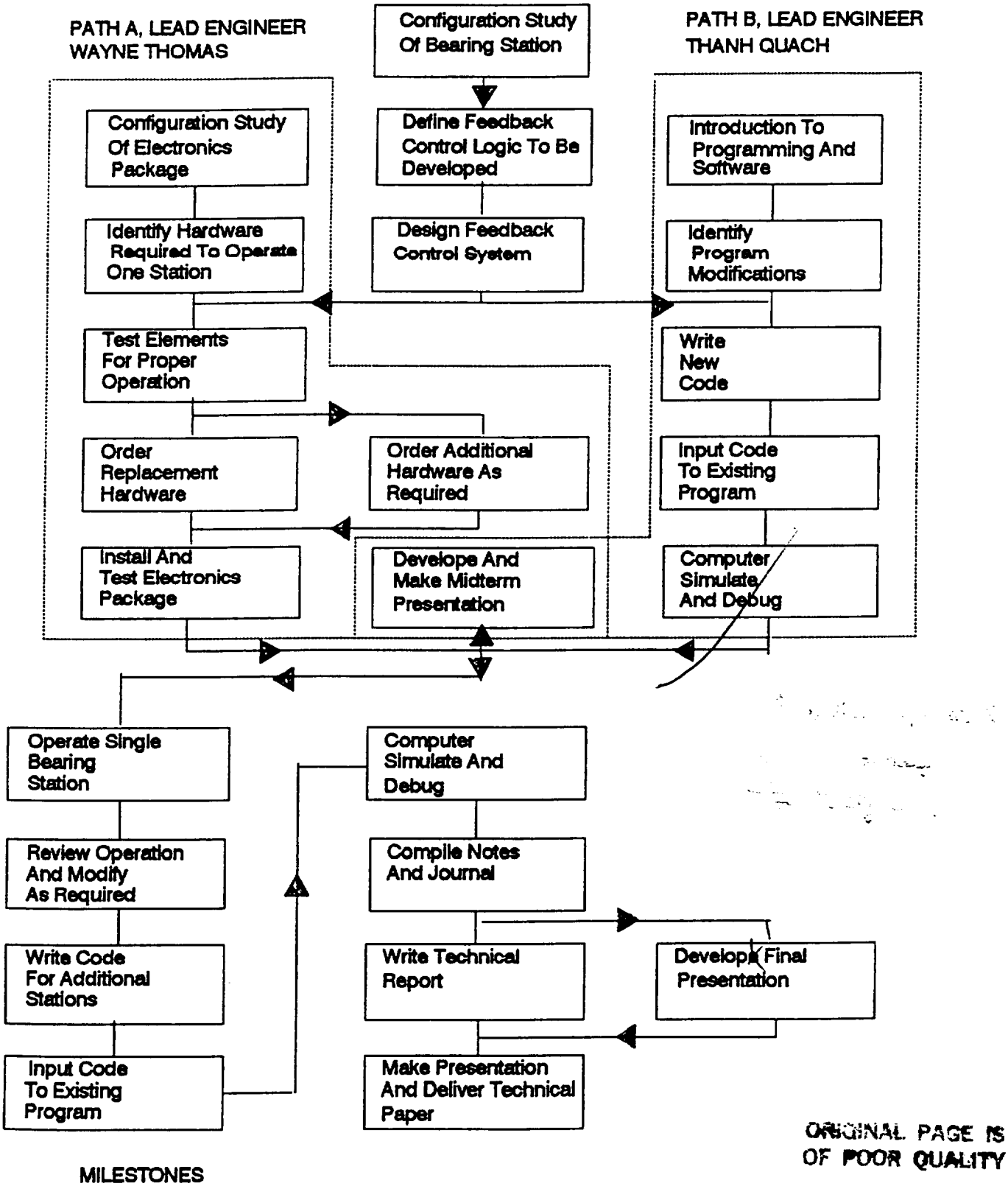
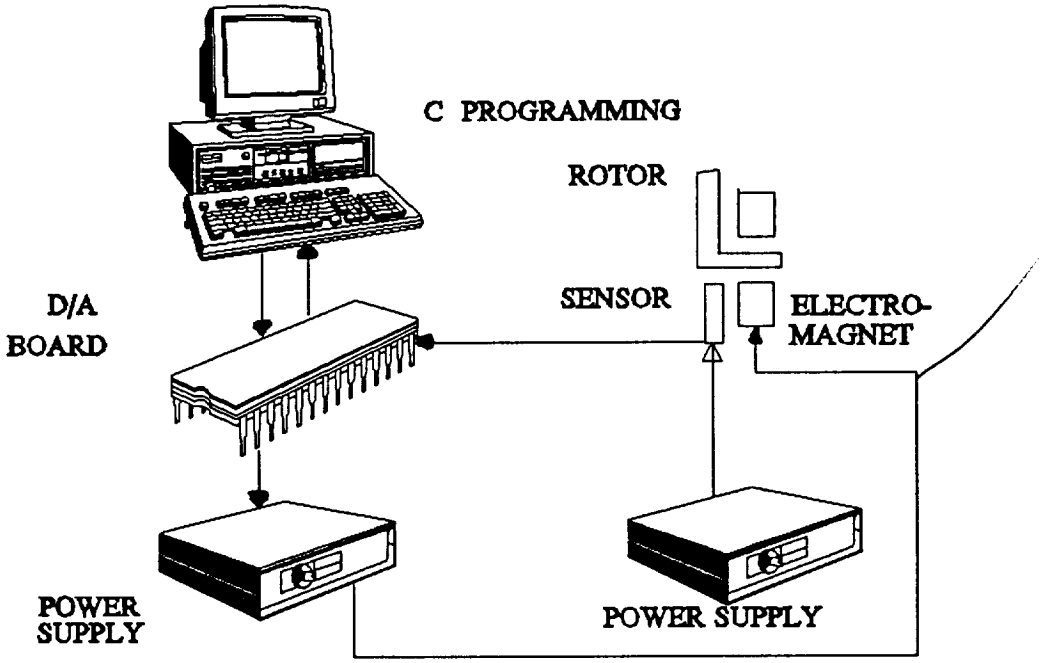


Figure 4a. Component Integration



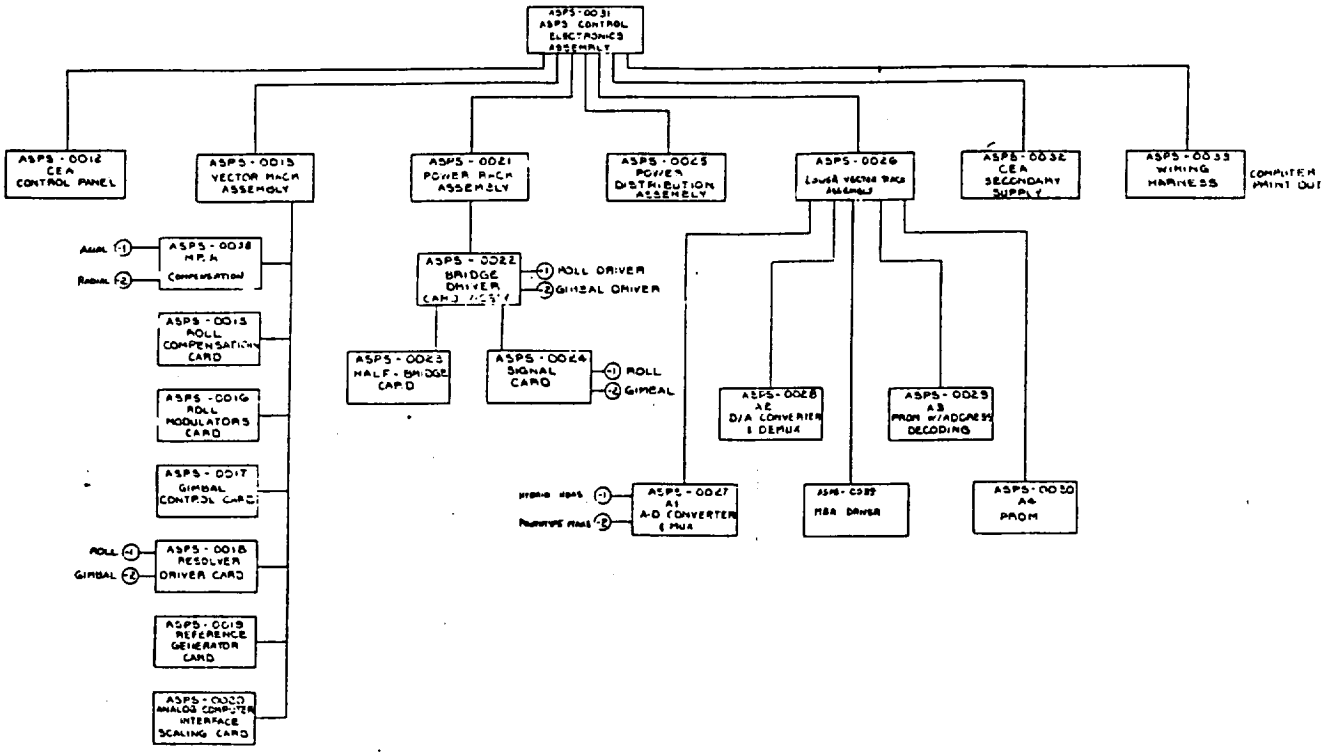
IV. HARDWARE DEVELOPMENT

1. Existing MBA Hardware Configuration.

The Magnetic Bearing Assembly (MBA) was initially developed to operate in a zero gravity environment. Sperry corporation performed the design and construction of the prototype assembly that we are currently modifying for operation in normal gravity. Sperry's approach was to use an analog feedback and control system in which most of the stabilization of the MBA was accomplished by electronic means, while an analog computer applied force commands and monitored operating parameters. According to Brian Hamilton of Sperry Corporation, considerable stability was achieved with this scheme in six degrees of freedom operation, but computer simulations indicated that instability would occur in actual operation due to vibrational and inertial effects posed by the carrier(19). Our concentration has been to incorporate a modern feedback control system to replace the existing hardware which was developed during the mid-seventies.

Before designing a feedback controller for the MBA an in-depth understanding of the existing hardware was needed. Technical documentation for the system is good as far as schematic diagrams are concerned, but it lacks the theory of operation for specific circuits. Essentially, while each circuit card has a designation (i.e. MBA Driver) which hints at its purpose, there is no detailed description as to how the circuit card actually works.

Figure 5. ASPS BreadBoard Electronics Family Tree, from Sperry Systems original design drawings, 1976.



The above situation was remedied by an in-depth study of the schematic diagrams coupled with a signal flow analysis. However, before the signal analysis could be performed the MBA had to be rendered operational.

Figure (5), on the previous page is an overall block diagram of the MBA electronic hardware. An initial assessment showed that +28vdc applied to the ASPS power distribution assembly was needed to energize the MBA assembly. Specifically, +28vdc was applied to terminal 1 and ground to terminal 12 of terminal block T107 (see figure 6, page 16). Initially this effort failed because a previous team had altered the ASPS-0012 CEA Control Panel (see figure 5, page 13) in an attempt to energize the MBA. Diodes D4, D36, and D45 had been desoldered and jumpered to other points within the Control Panel (see figure 7, page 17). Apparently, this was done to alter the application of operating voltages to circuit cards so that elements of the MBA could be operated individually. After removing the jumpers and reconnecting the diodes the MBA was successfully turned on.

It is interesting to note that existing hardware is designed to accept an input of +28vdc. This was done because +28vdc is a common output from power supplies used in aerospace applications. The power supply delivered with the MBA was inoperative and no schematics of it were included in the documentation provided by NASA. This power supply is currently under repair by Old Dominions Universities electronics shop. In its' place we are

using a KEPCO DRC-40 power supply in place of the original. This power supply is capable of providing 40 amps of current and must be used with caution because the MBA only requires a fraction of this power.

Figure 6. Power Distribution Assembly Schematic, ASPS-0025 from Sperry Systems final design drawings, 1977.

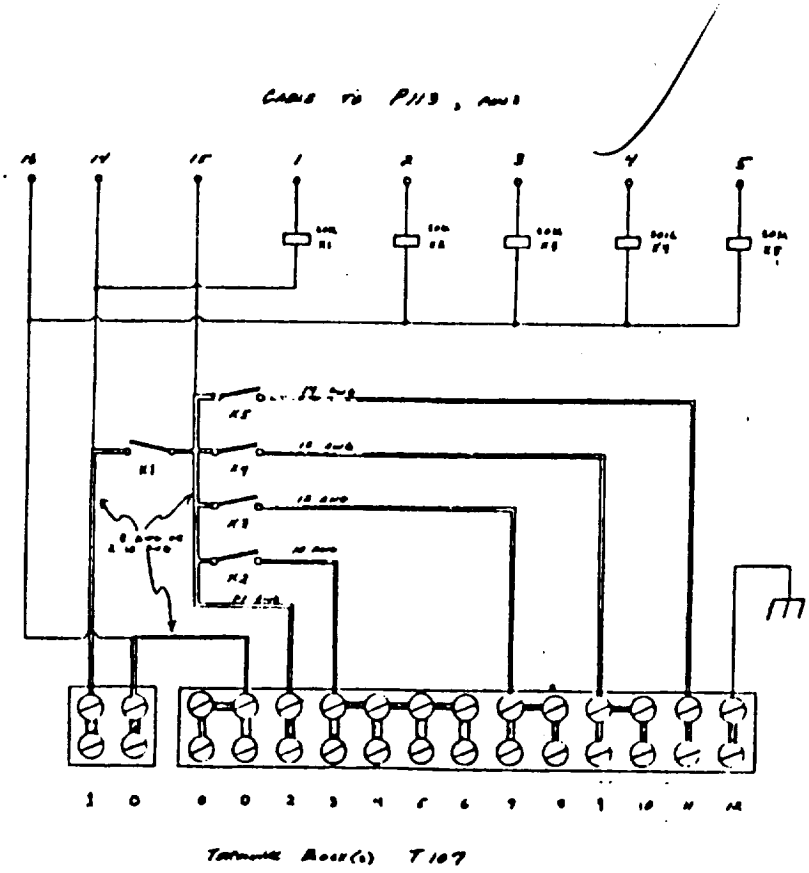
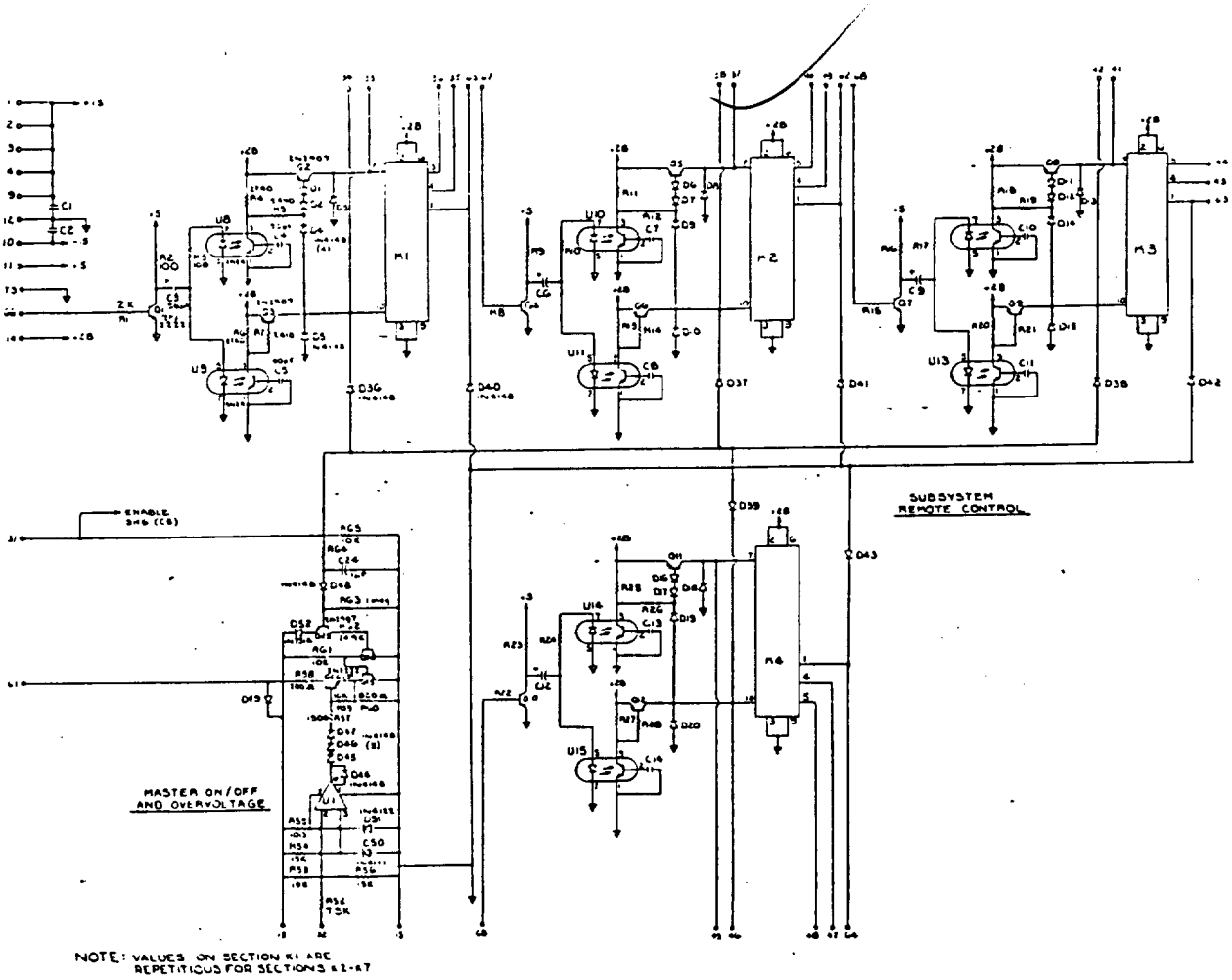


Figure 7. Control Electronics Assembly, ASPS-0012 from Sperry Systems final design drawings, 1977.



During normal operation, in which the rotor disc was attached to the top magnetic bearings we observed a current drain of only 4 amps on the front panel meter of the DRC-40 power supply.

Once power was applied to the MBA it was possible to perform a signal analysis of the circuitry to determine the exact function of the cards relative to the feedback control system detailed in the final design review (ASPS final design review, May, 1977). By comparing this document with the schematic diagrams, the following subsystems were initially determined necessary for operation with a digital feedback control design:

1. Power supplies (+28vdc, +,-15vdc, +,-5vdc)
2. Position Sensor Amplifiers (ASPS-0036)
3. MBA Driver (ASPS-0039)
4. Axial MBA Compensation (ASPS-0038-1)

With the above circuit cards identified for operation of the MBA a detailed analysis was conducted. This analysis allowed us to learn just how the MBA operated in its current configuration, while at the same time determining if any components were faulty.

One of the first components tested for proper operation was the CEA Secondary Supply, dwg.ASPS-0032 (see figure 5, page 13). This is the power supply which converts the +28vdc input from the KEPCO DCR-40 power supply into the positive and negative 5 and 15 volts needed by the transistors and integrated circuit chips. The outputs are applied to bus-bars on the back of the Power Rack Assembly, dwg. ASPS-0031 (Figure 8, page 20). These bus-bars (T103 on figure 8)also distribute the +28vdc that is used to apply

voltage to the DC magnets. The actual test involved simple voltage measurements along the bus-bar and reading the test points.

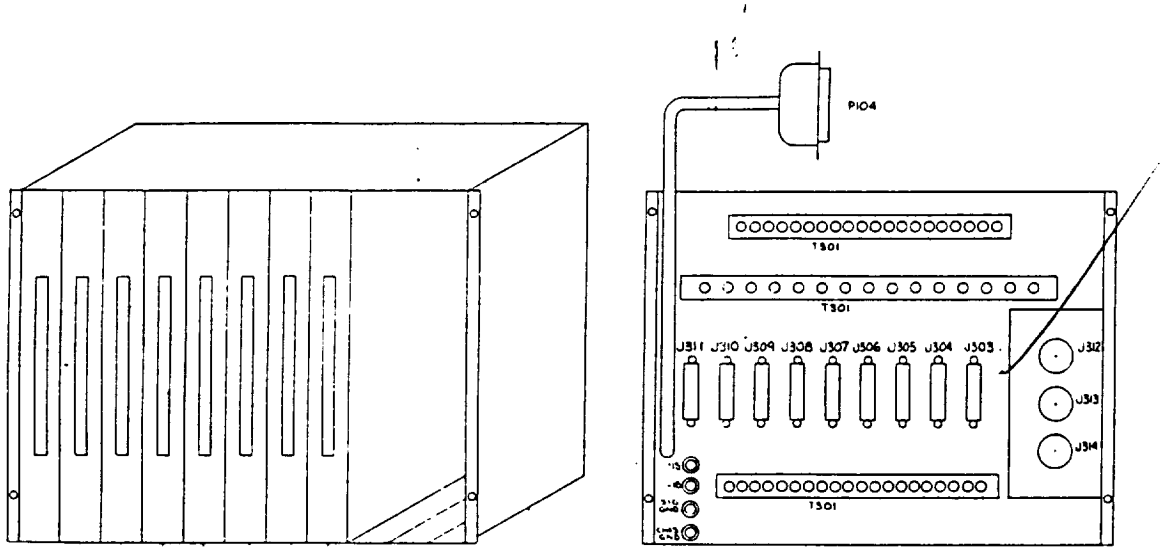
The next elements that were tested were the position sensors. The outputs from all five bearing stations can be read off of the front panel of the CEA control panel (figure 10, page 24). Readings were taken from gaps A, B, and C which are the outputs from the radial gap sensors. The following results were obtained for the upper (rotor assembly stuck to the top magnets) and lower (rotor assembly stuck to the bottom magnets) limits of rotor travel:

Position	Gap A	Gap B	Gap C	Gap Width
Top	1.47vdc	1.45vdc	1.44vdc	+7.315E-3m
Bottom	-1.48vdc	-1.42vdc	-1.46vdc	-7.315E-3m
Range	403.27V/m	392.33V/m	396.44V/m	**********

Table 1. Position Sensor Output

Figure 8. Power Rack Assembly, ASPS-0031,

from Sperry Systems final design drawings, 1977.



Further analysis of the position sensor circuitry was made to gain a better understanding of how the feedback control system operates. Figure 9, page 23, is a simplified schematic of one of the six position amplifiers. The position sensors are used to convert the mechanical position of the MBA rotor into a varying DC signal. The current gap width is 0.288 inches for all axial bearing stations. The input from the position sensors to the position sensor amplifiers is a 25 Mhz sine wave that varies from 7 -10 vpp. This variation is dependent on the relative position of the MBA rotor. As the MBA nears the top sensor, the signal at AA on figure 9 approaches 10vpp while the signal at BB (from the bottom position sensor) approaches 7vpp. This circuit functions as a differential amplifier with a gain less than unity. For instance, with the AC voltage inputs described above, the output at CC is approximately +1.5vdc(this circuit not only functions as a comparator but also converts the AC input into a DC output).

Finally, the MBA Driver cards, drawing ASPS-0039 (Figure 10, page 24) and the Axial MBA Compensation cards, drawing ASPS-0038-1 (Figure 11, page 25) were investigated. Through extensive signal analysis it was determined that these cards function as the heart of the analog feedback control system. Correlation with the final design review depiction of the MBA Driver Module block diagram (Figure 12, page 26) substantiated our suspicions. Specifically, the Axial MBA Compensation circuit cards are represented by the elements to the left of the dashed line on figure 12, while the MBA Driver cards are represented by the remaining portion to the right

of the dashed line. The lower portion titled "Part of VEA" on figure 12, on the previous page represents the position sensor circuitry whose final output (CC on figure 9, page 23) is designated as Δg.

Figure 9. Detail of Position Sensor Amplifiers, ~~produced by the author~~.

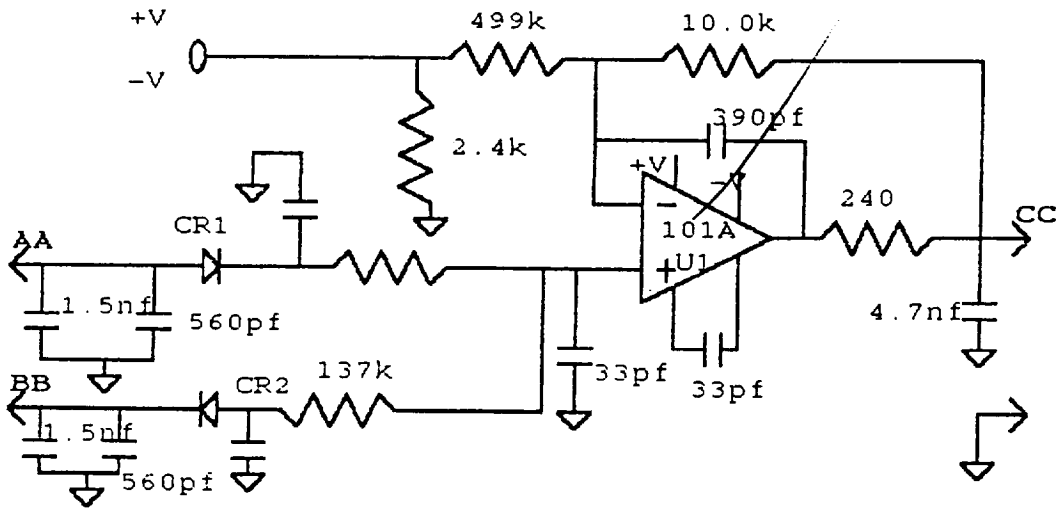


Figure 10. MBA Driver, ASPS-0039, Sperry Systems Final Design Drawings, 1977.

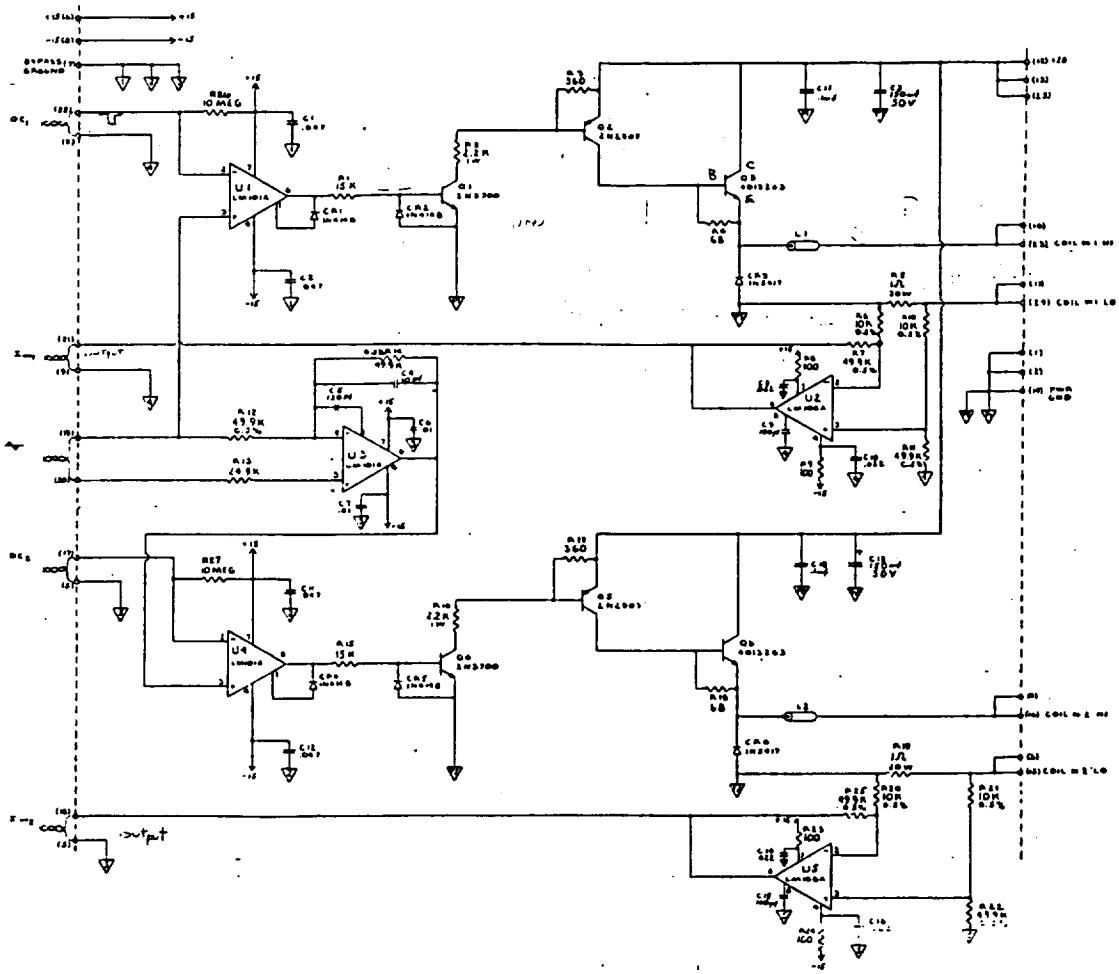


Figure 11. Axial MBA Compensation, ASPS-0038-1, Sperry Systems Final design drawings, 1977.

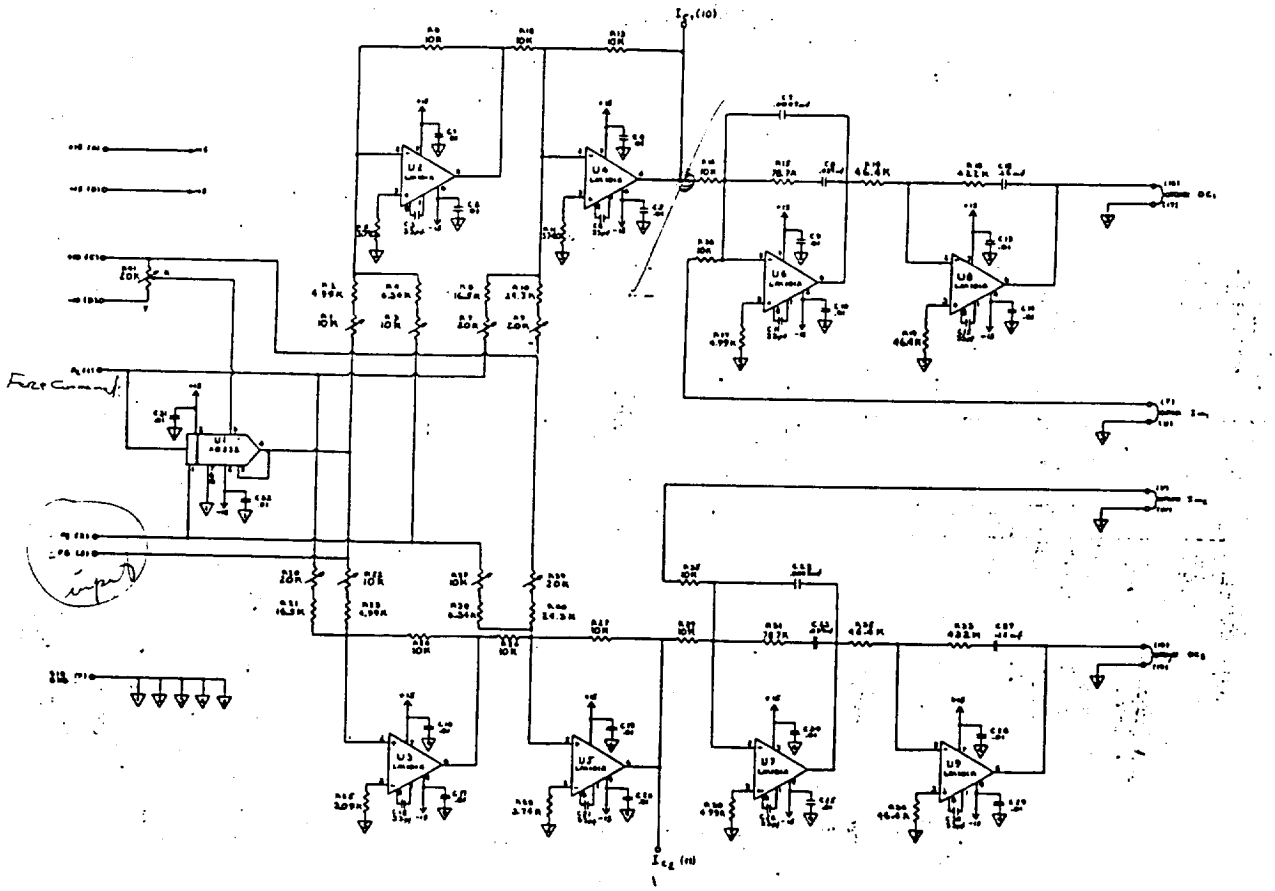
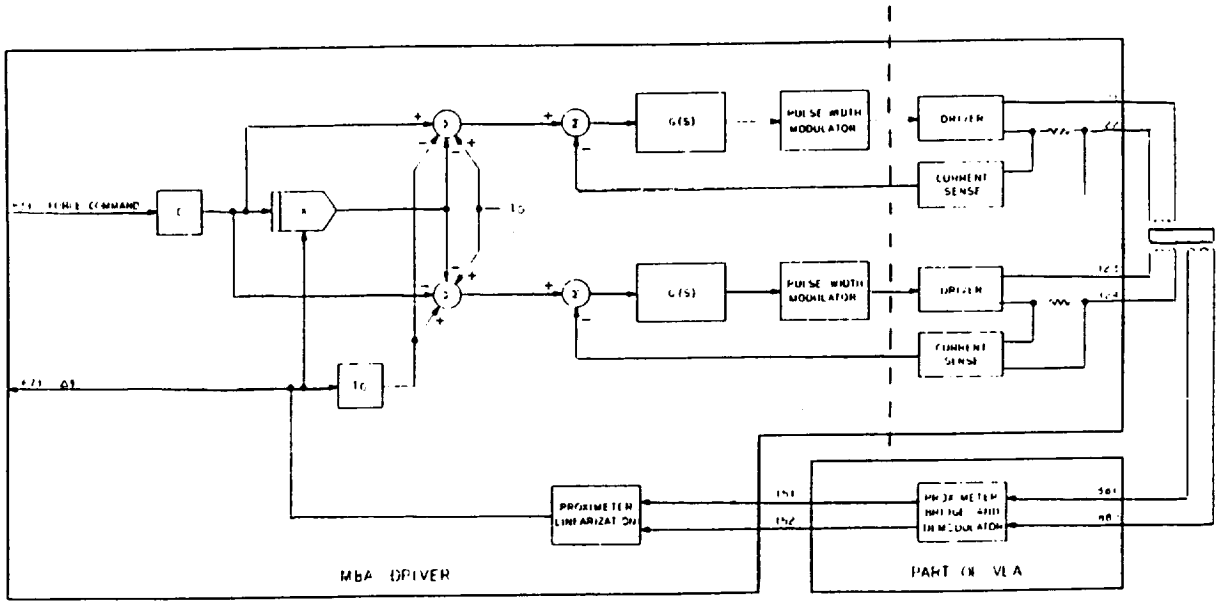


Figure 12. MBA Driver Module Block Diagram, ASPS Final Design review, page 5-21.



2. Goals of Hardware Development

The initial goal of the hardware development, which was to learn how the existing system operates, has been met. After initial repairs and the powering up of the entire system a full understanding of its operation has been gained. Much of the circuitry has not been discussed because most of it was designed to handle roll, yaw and pitch of the rotor assemble under actual operating conditions. Again, this is an extensive analog feedback and control system designed to operate with a 1970's vintage analog computer. In light of this design teams approach to minimize hardware and incorporate a digital computer into the control scheme as a more modern approach, this hardware will not be needed. Testing has shown that the MBA Driver circuit cards can be removed from the MBA and yet the position sensors still operate properly. This will allow the use of off the shelf hardware to apply drive currents to the DC magnets while using the position sensor outputs from the front panel of the CEA control panel.

3. Hardware Design Approach

The hardware analysis has shown that it is impractical to tie a digital computer into the existing feedback control system. Several attempts were made to apply force commands that would vary the current to the DC magnets. This attempt failed because of the absence of the original analog computer that would have closed many loops in the feedback control circuitry. Moreover, our current goal is to achieve stable magnetic levitation with one bearing station in one degree of freedom. The existing system was designed to

operate in zero gravity with both the upper and lower banks of DC magnets in operation.

No I/we/us.

Our approach will only require the use of the upper banks of magnets to compensate against the force of gravity. With this in mind we have experimented with the use of a KEPCO BOP 20-20M programmable power supply. This unit accepts inputs from a digital computer to adjust its current output and also provides a current feedback signal. We have already reconfigured the MBA circuity so that the programmable power supply will replace the MBA Driver module for bearing station A. This was done by manufacturing a connector which applies current at pins 10 and 23 and accepts a return at pins 11 and 24 of MBA Driver ASPS 0039 (Figure 10, page 24) Experimentation has shown that by increasing the current input towards the limit of 1.4 amps, the bearing rotor can be drawn from the center position to stick to the top magnet.

4. HARDWARE DEVELOPMENT RESULTS.

The KEPCO BOP 20-20M programmable power supply proved more than adequate as a current source to drive the electro-magnets in the Magnetic Bearing Assembly. The fast transient response (83μsec) was more than adequate for stable operation of the magnetic field. One improvement made over the initial design is the installation of a one ohm, 25 watt resistor on the return line from the MBA coil. This resistor serves to dissipate energy resulting from the rapid changes in current during MBA operation. Effectively, as the coil current decreases the resistor bleeds off this energy in the form of heat. This resistor has cooling fins

attached and normally gets hot during operation.

The KEPCO BOP 20-20M power supply has a gain of 2 and is capable of a 20 ampere current output(this would correspond to a 10 volt input to the programming input terminals). If this amount of current was applied to the MBA electro-magnets, it is likely that the windings in the magnets would fuse. For this reason, the current limits on the power supply have been set to roughly + or - 1.75 amps. This allows operation of the DC magnets slightly beyond their saturation point, but ensures that damaging currents can't be applied. Additionally, this allows for lengthy operation of the bearing assembly without overheating the DC magnets.

In addition to the installation of the programmable power supply, a suitable wiring harness was developed. All of the connections in the harness are soldered to reduce the possibility of noise generation due to loose connections (this would result in instability during operation). Additionally, a connector was installed so that the power supply output can be applied to any bearing station by simply removing the associated MBA driver card and attaching the connector. During early testing this was done quite frequently to avoid overheating the DC-magnets (before operating current limits were established).

To operate the current hardware configuration, first turn on the primary power supply at the bottom of the equipment rack. This power supply should indicate +28vdc during normal operation (make adjustments if needed). Next, turn on all of the toggle switches on the front of the ASPS control panel (the associated

amber and green lights should illuminate). This applies power to the entire MBA, and will allow the gap sensors to operate. Now, turn on the KEPCO programmable power supply and position the current control switch in the on position while ensuring that the voltage control switch is in the off position. At this point the system is fully operational and is ready to interface with the digital computer.

5. FUTURE HARDWARE DEVELOPMENT.

The current hardware configuration provides stable operation for a single bearing station but can easily be improved and expanded to operate the entire MBA. Two alternatives are available to achieve this end. One alternative is to expand the existing configuration by installing additional KEPCO programmable power supplies (a total of 5 are needed). The KEPCO BOP 50-2M is an ideal candidate for this application. This model provides a current output of + or - 2 amperes as opposed to the 20 amperes provided by the current model. The advantage here is that the BOP 50-2M model only costs $1,174.00 (less than half the price of the BOP 20-20M) and occupies only half the space.

The physical connection of five power supplies into the existing system would be relatively easy, but integration into the digital control scheme would require additional electronic development. Specifically, the current A/D & D/A convertor (DT2811) only provides one output while five are needed to fully operate the MBA. This can be incorporated by developing electronics that are capable of multiplexing the output signals

among the five power supplies. For this scheme to work a memory device would be needed at each power supply to maintain its current level while the software steps among the five bearing stations. A simple commercial memory module could easily be adapted to accomplish this end.

A second alternative is to modify the existing MBA driver cards to accept digital input. This can be accomplished by replacing U1 (see fig. 10, page 24) with a non-inverting, low gain operational amplifier configuration. Additionally, resistors R6, R12 and R10 need to be desoldered to disconnect the feedback loop for the analog control system. With the proper gain configuration on the new operational amplifier (preferably a LM-741 op-amp) analog inputs from the DT2811 patch panel could be provided to pins 19 and 22 of the MBA driver (this input is currently connected to the programming input of the programmable power supply).

Either of the above modifications can be incorporated to expand the present system to operate the entire MBA. In either case a parallel software development will be required. Overall, the hardware development accomplished all of its objectives.

V. Software Development

1. Software Background

According to NASA's final report on "The Development of the ASPS Vernier System" (publication 17-1829), Sperry Flight Systems designed and successfully tested a six degree of freedom model of a magnetic suspension system. The software associated with the ASPS control system was developed on an analog computer of the 1970's. This ASPS assembly designed for an analog controller was tested through a computer simulation. No documentation is available on the computer simulation approach. In addition, no existing documentation mentions any other applications of the microcomputer in relation to the ASPS system.

2. Software Goal

In conjunction with the overall objective of this project, the goal of the software division is to produce a program in the C language for the analysis and control of a single bearing station.

3. Approach

To achieve the above goal, much time was invested to learn the C programming language as well as the C compiler. In addition, an understanding of the new timer board, the existing digital to analog board, and the interface process and subroutines was necessary. Despite the above obstacles, the approach for achieving the software goal consists of the following:

1) identifying the required changes to an existing program, which has applicable oscilloscope capabilities (developed by Mehran Ghofrani). Reference report.

2) developing test programs to test the interface and applicability of the DT 2811 analog to digital converter board and the timer board.

3) writing and debugging a C program for the single magnetic bearing system.

With time permitting, the single bearing program will be modified to reflect the existing three bearing system.

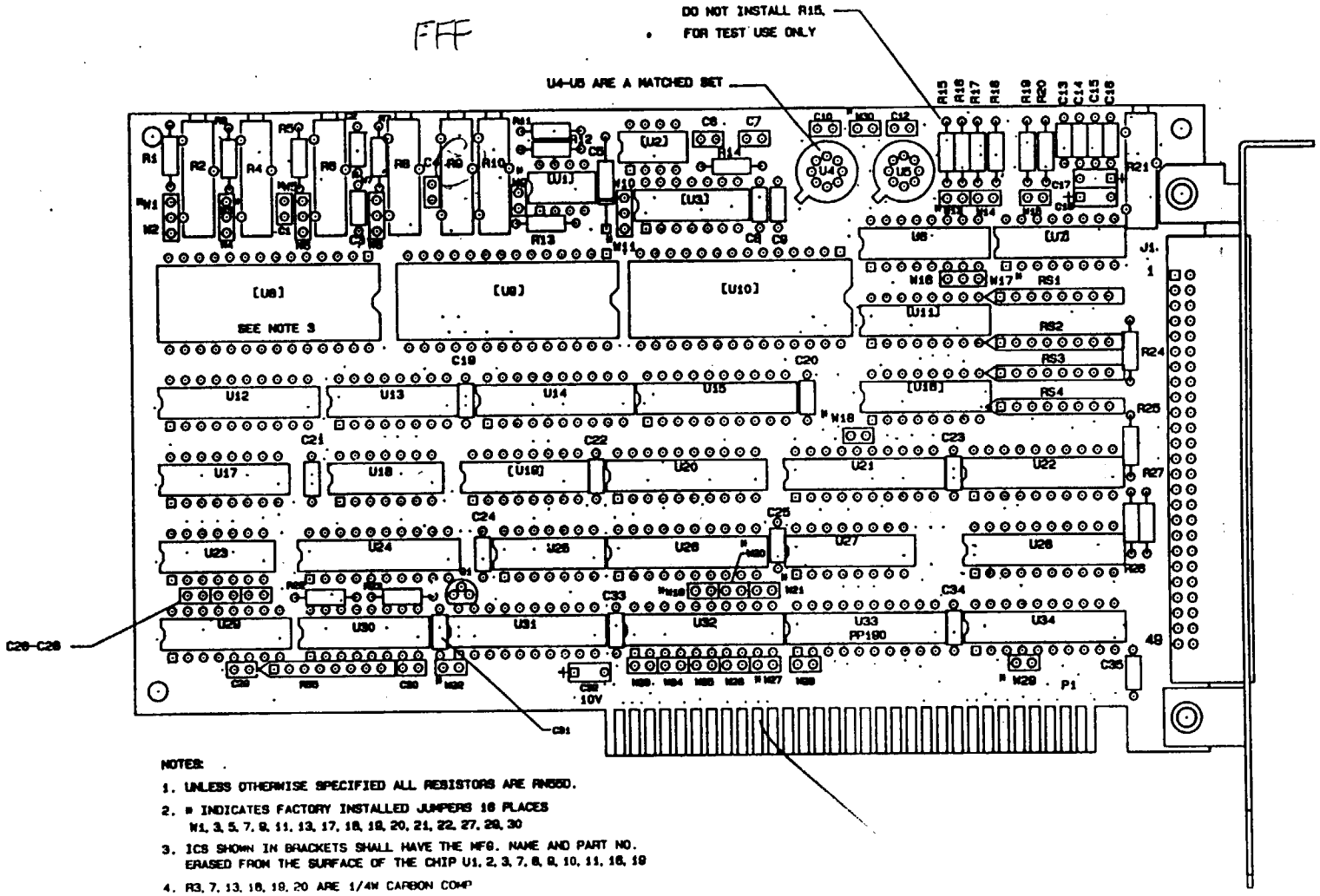
4. Hardware & Software Configuration for Computer System

The above approach required an understanding of both the existing hardware and software configurations of the computer system. Knowledge of how the interface occurs between the data acquisition board, the timer board, and the Microsoft quick C software package will enable the production of the levitation program.

Data Acquisition (DAC) Board

The DAC board used for the ASPS project is Data Translation's DT2811, which operates from the +5V, +12V, and -12V power lines on the IBM PC bus. It is a low-cost analog and digital I/O board designed for use with the IBM compatible microcomputer systems, which in our case is the Gateway 2000. This half-size board features three subsystems, each performing distinct functions: analog to digital conversion (A/D), digital to analog conversion (D/A) and digital I/O. The DT 2811 (DWG 07237)

FFF



NOTES:

1. UNLESS OTHERWISE SPECIFIED ALL RESISTORS ARE RN55D.
2. # INDICATES FACTORY INSTALLED JUMPERS 16 PLACES
   W1, 3, 5, 7, 9, 11, 13, 17, 18, 19, 20, 21, 22, 27, 29, 30
3. ICS SHOWN IN BRACKETS SHALL HAVE THE MFG. NAME AND PART NO. ERASED FROM THE SURFACE OF THE CHIP U1, 2, 3, 7, 8, 9, 10, 11, 16, 19
4. R3, 7, 13, 16, 19, 20 ARE 1/4W CARBON COMP
5. R22 and Q1 are not installed on PGH.

		DATA TRANSLATION MARLBORO MASSACHUSETTS 01752
N/A	N/A	USER MANUAL ASSEMBLY DT2811
		07237 A

typically plugs into any one of the fully bussed expansion slots in the compatible computer backplane. Highlights of the main features of the DT2811 are depicted in figure 13.

In order to interface with the input/output signals, the DT707, a simple screw terminal panel, provides connection points to the interface. User connections are made through convenient screw terminals which are labeled in silkscreen (DWG 00817). The connection to the host computer system is made via an integral 50-conductor flat ribbon cable (3.3 ft) which plugs directly into the ST2811 boards jumper connector. However, this same DAC board is accessed via a 10-bit address received from the host processor. The A/D conversions are initiated with a either a software or hardware trigger. Even though the hardware trigger (which consists of an externally generated electrical pulse) is possible, the software trigger (which is issued by the program running board) is more convenient. The interfacing subroutine librarys are incorporated into the LPCLAB software package. This real time software facilitates A/D and D/A operations through commands in several languages including C. Interface is accomplished by loading the A/D gain and channel registers with the valid gain and channel numbers along with calling the correct variable and function names, depending on the desired application or operation.

Figure 13

TABLE 1-1: MAIN FEATURES OF DT2811

MODEL	A/D RANGE	A/D RES	A/D GAIN	A/D MAX THRUPUT	D/A RANGE	D/A RES
DT2811-PGH	0-5V ±5V ±2.5V	12	1,2, 4,8	20kHz 20kHz	0-5V ~~±5V~~ ±2.5V	12
→ DT2811-PGL	0-5V ±5V ±2.5V	12	1,10, 100, 500	20kHz 2.5kHz 2.5kHz	0-5V ±5V ±2.5V	12

NOTE:
- RES = Resolution in bits.

---

* Throughout this manual, the designation DT2811 is used to refer to the two boards, DT2811-PGH and DT2811-PGL. The full designation (DT2811 with suffix PGH or PGL) is used when referring to specific models.

Timer Board

• The DT2819 timer board is a multi function counter/timer which provides counting, sequencing and timing functions for data transfers between the host compatible microcomputer systems and a peripheral device such as the DAC board. This timer board plugs into a single I/O expansion slot in the IBM backplane and operates from the +5V power line on the IBM bus. The counter/timer subsystem consists of software selectable 5MHz or 1MHz base frequencies and an AM9513A system timing controller. The AM9513A contains five general purpose counters that use scaled frequencies from the 5MHz or 1MHz clock, or from an external source connected to the DT2819. These counters are software configurable for active-high or low polarity and for counting up or down in BCD or binary. They can also be cascaded to form an effective counter with a length of up to 80 bits. In this way, the host processor may read an augmented count at any time without disturbing the counting process. In addition, the DT2819 provides a single interrupt that is configurable for levels 2-7. This interrupt is generated on a high to low transition signal from an external source connected to the board. The timer board is interfaced through eight 8 bit I/O mapped registers that provide all communication with the Am9513A chip. More information on the Am9513A system timing controller chip is depicted in figure 2-362. In order to program this chip, the PACER timer board software, which contains the timer board subroutines, is necessary. Similar

Figure 2-362

Am9513A/AmZ8073A

System Timing Controller

DISTINCTIVE CHARACTERISTICS

- Five independent 16-bit counters
- High speed counting rates
- Up/down and binary/BCD counting
- Internal oscillator frequency source
- Tapped frequency scaler
- Programmable frequency output
- 8-bit or 16-bit bus interface
- Time-of-day option
- Alarm comparators on counters 1 and 2
- Complex duty cycle outputs
- One-shot or continuous outputs
- Programmable count/gate source selection
- Programmable input and output polarities
- Programmable gating functions
- Retriggering capability
- +5 volt power supply
- Standard 40-pin package

GENERAL DESCRIPTION

The Am9513A System Timing Controller is an LSI circuit designed to service many types of counting, sequencing and timing applications. It provides the capability for programmable frequency synthesis, high resolution programmable duty cycle waveforms, retriggerable digital one-shots, time-of-day clocking, coincidence alarms, complex pulse generation, high resolution baud rate generation, frequency shift keying, stop-watching timing, event count accumulation, waveform analysis, etc. A variety of programmable operating modes and control features allow the Am9513A to be personalized for particular applications as well as dynamically reconfigured under program control.

The STC includes five general-purpose 16-bit counters. A variety of internal frequency sources and external pins may be selected as inputs for individual counters with software selectable active-high or active-low input polarity. Both hardware and software gating of each counter is available. Three-state outputs for each counter provide pulses or levels and can be active-high or active-low. The counters can be programmed to count up or down in either binary or BCD. The host processor may read an accumulated count at any time without disturbing the counting process. Any of the counters may be internally concatenated to form any effective counter length up to 80 bits.

The AmZ8073A* is functionally equivalent to the Am9513A with timing enhancements which allow it to be fully speed compatible with the AmZ8001 and AmZ8002 microprocessors.

BLOCK DIAGRAM

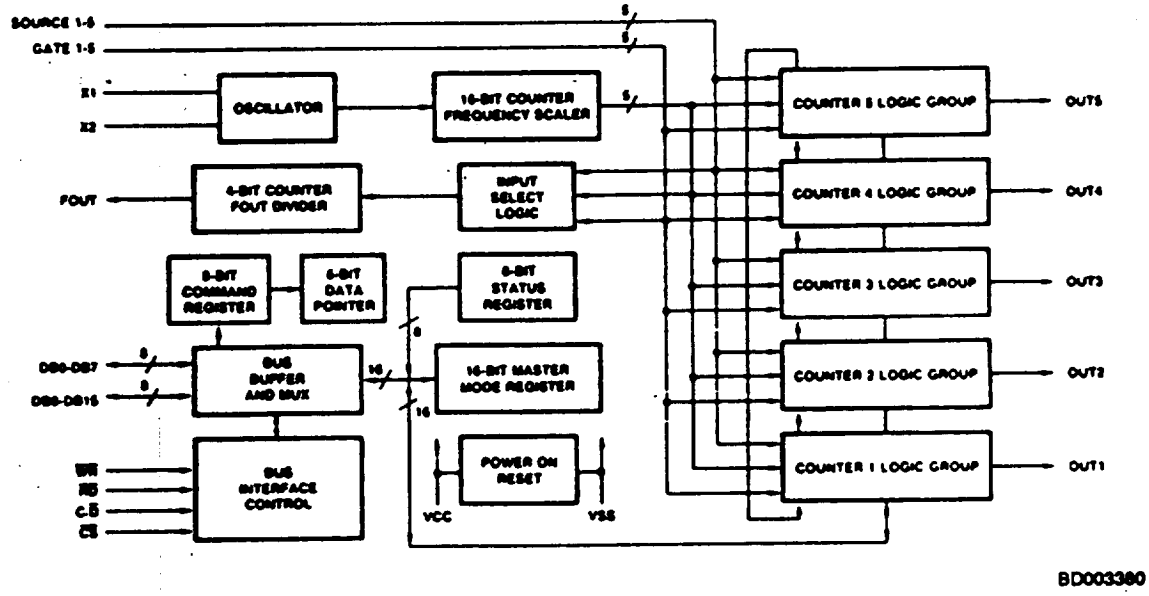


Figure 1-1.

*Z8000 is a trademark of Zilog, Inc.

01731D

37A.

to the DAC board, the desired interface operation is performed by calling the appropriate subroutines from PACER and providing the correct variables associated. Fortunately, the PACER subroutines are callable from Microsoft Quick C.

#5 **Microsoft Quick C**

Microsoft Quick C is a simplified version of Microsoft C. As the name suggests, it is produced by Microsoft. For the purposes of this project, Microsoft Quick C version 2.5 is used. This version required 448 kilobytes of available memory. Some special features of this version are that it has an increased ANSI standard C compatibility, a new quickwatch debugging feature and a color window support which can be customized. The software has been installed for compatibility with a small memory model. This means that 64K is allotted as the code segment limit and 64K as the data segment limit. It is worth noting that certain features of QuickC must be changed from the default when a program wishes to incorporate the DAC or timer boards. In the case of the DAC board, the stack size needs to be increased when continuous data conversion operations are desired. Also, if the timer board were utilized by a program, the mapping feature under the option, as well as the stack check need to be changed when timer board interface and an interrupt call is desired.

5. Problems Encountered and Modifications

In our attempt to achieve the software goal of the project, several problems were encountered. The first problem encountered was the software incompatibility problem. The problem

was that the original LPCLAB version 1.0 was not compatible with QuickC 1.0. This fact was not realized until later in the software development program since some LPCLAB commands were executable while others were not. Also, since the timer board software (PACER) required QuickC 2.5, both the QuickC 2.5 and the LPCLAB 2.01 were purchased and installed soon after. After the installation of the correct version of the QuickC compiler, the memory setup for the QuickC compiler and the LPCLAB was found to be incorrect. This was due to QuickC detecting a mismatch of the memory library files with those used by the LPCLAB setup requirement. This problem was resolved by reinstalling the QuickC and LPCLAB for a small memory model. Next, test programs such as the one in figure 14 were developed and successfully interfaced with the DAC board. However, the interface with the timer board in the test programs was unsuccessful. This was probably due to the improper call of the PA_SETUP_INTERRUPT or other required parameter definitions. Since the interface with the timer board was unsuccessful, the program development only utilized the DAC board operations to control the electromagnetic levitation process.

6. Program Development Analysis

Ghofrani Program Modifications

The Ghofrani program is a complex and very user friendly program that controls the leviation of a five degree of freedom magnetic suspension system. This program starts off by defining the interrupt service routine (ISR) by using DOS assembly interrupt commands. These commands were changed to the timer board

interrupt commands. The controller loop provides for filtering of both the yaw changes as well as the lateral changes. The only aspect that was modified was the controller model used. This was changed from a PID to a PD control model. Other changes to the Ghofrani program included the dual phase advance compensator aspects, the timer setup aspect, the matrix manipulation and decoupling aspects, and the data acquisition I/O aspects. A copy of the original program is included in the appendix with the changes highlighted.

Single Magnetic Bearing Assembly Levitation Program

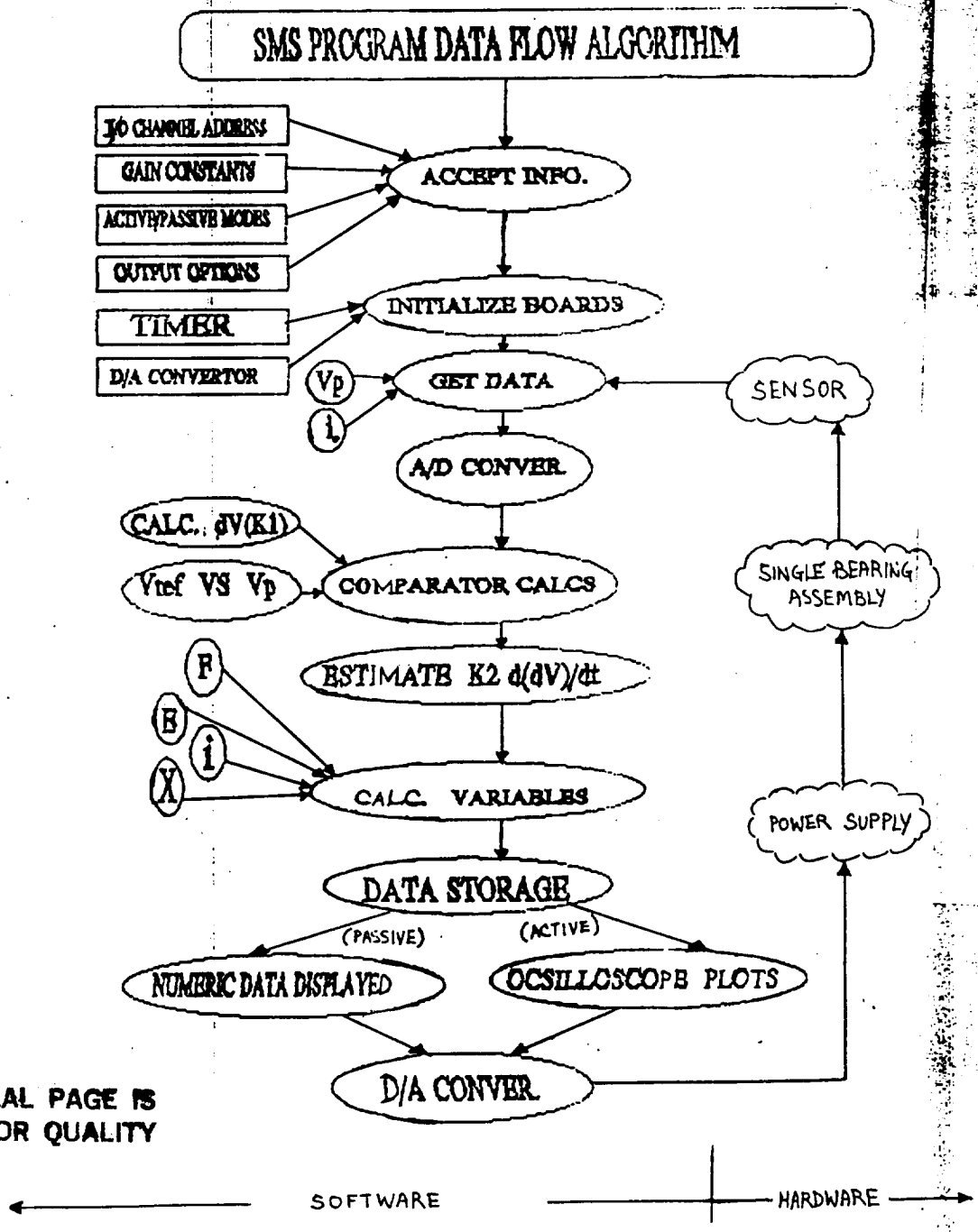
This program was developed for a one degree of freedom suspension system, according to the program algorithm as depicted in figure 15. The PD control section of this program was modelled after the test program 'levate' (fig 14). It starts off with the main menu, where the user is given three choices; passive operation, active operation, or quitting. The passive mode entails running the levitation control without any screen interaction whereas the active mode entails running the control with the oscilloscope functions. In either case the voltage data retrieved through the DAC board is written to a file until the control loop is broken when any key on the keyboard is hit. The data stored in the SBAVOLT file can be called later for other calculations such as the current, force, and position. These calculations are executed and then written into different files, once the control is terminated. It is worth noting that the equations for the optional calculations have not been developed. Therefore, this section only

has debugging statements for testing purposes. Finally, the user is returned to the main menu and may reinitiate the control routine by either choosing the active or passive keywords. A copy of the program is included in the appendix.

Figure 14. Sample Test Program

```
#include<stdio.h>


void main(void)
{
float intergral,xold,speed;
short xvalue,actcontrols,control;
int b,n,m;
LPINIT();
LPSB(1);
intergral=0;
b=0;
xvalue=0;
actcontrols=0;
intergral=0;
while(b<15000){
/*while(m<5000){
m=m+1;
} */
xold=xvalue;
LPAV(1,1,&xvalue);
printf("%i\n",xvalue);
xvalue=xvalue-2047;
intergral=intergral+.01*xvalue;

speed=(xvalue-xold)/.01;
control=-.5*xvalue-.005*speed-0*intergral;
control=control+2314;
LPDV(0,control);
b=b+1;
}
control=2047;
LPDV(0,control);
}
```

Figure 15. Data Flow Algorithm



VI. FEEDBACK CONTROL DESIGN AND ANALYSIS

1. BACKGROUND

The controls model originally developed for the ASPS represents a mathematical model of a PID controller. The complexity of the original model evolved from the integration of a six degree of freedom system with extensive control electronics (analog system). Figure 16 shows the original control logic.

2. CONTROL LOGIC GOAL

The objective is to design a mathematical control model that represents the stabilization of a single bearing station.

3. APPROACH

This aspect of the project first focused on choosing a type of control that was compatible with the magnetic rotary joint application. The options of a proportional-derivative (PD) and a phase advance control were investigated and the PD controller was chosen. Reasons for this decision included that the PD control is a simple regulating processing controller, and an error minimizer. A general representation of the PD controller under inspection is included as figure 17. Upon further investigation, the following top level diagram of the controller relation was developed (Figure 18).

Figure 16. Original Control Logic

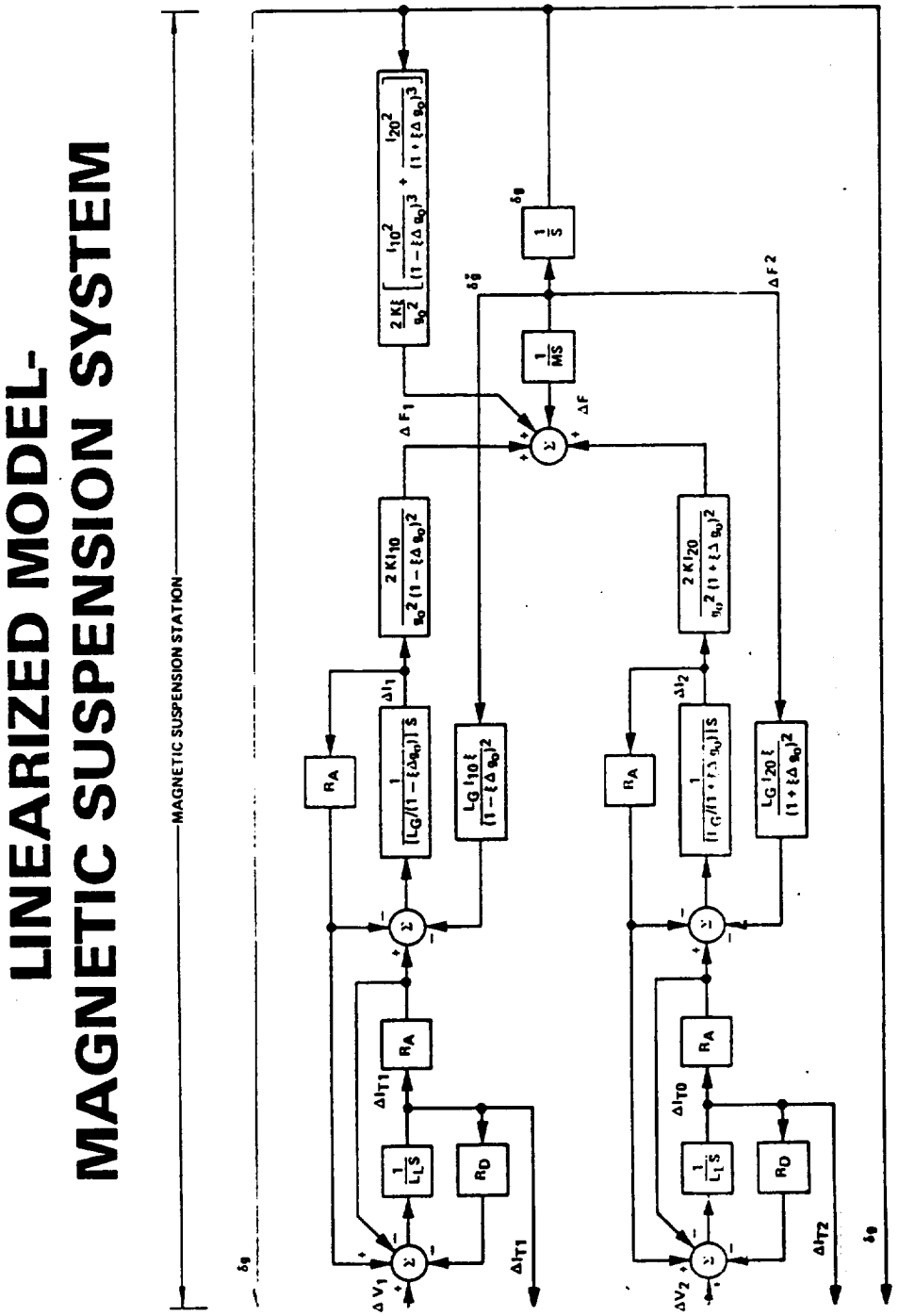


Figure 17. General PD Control

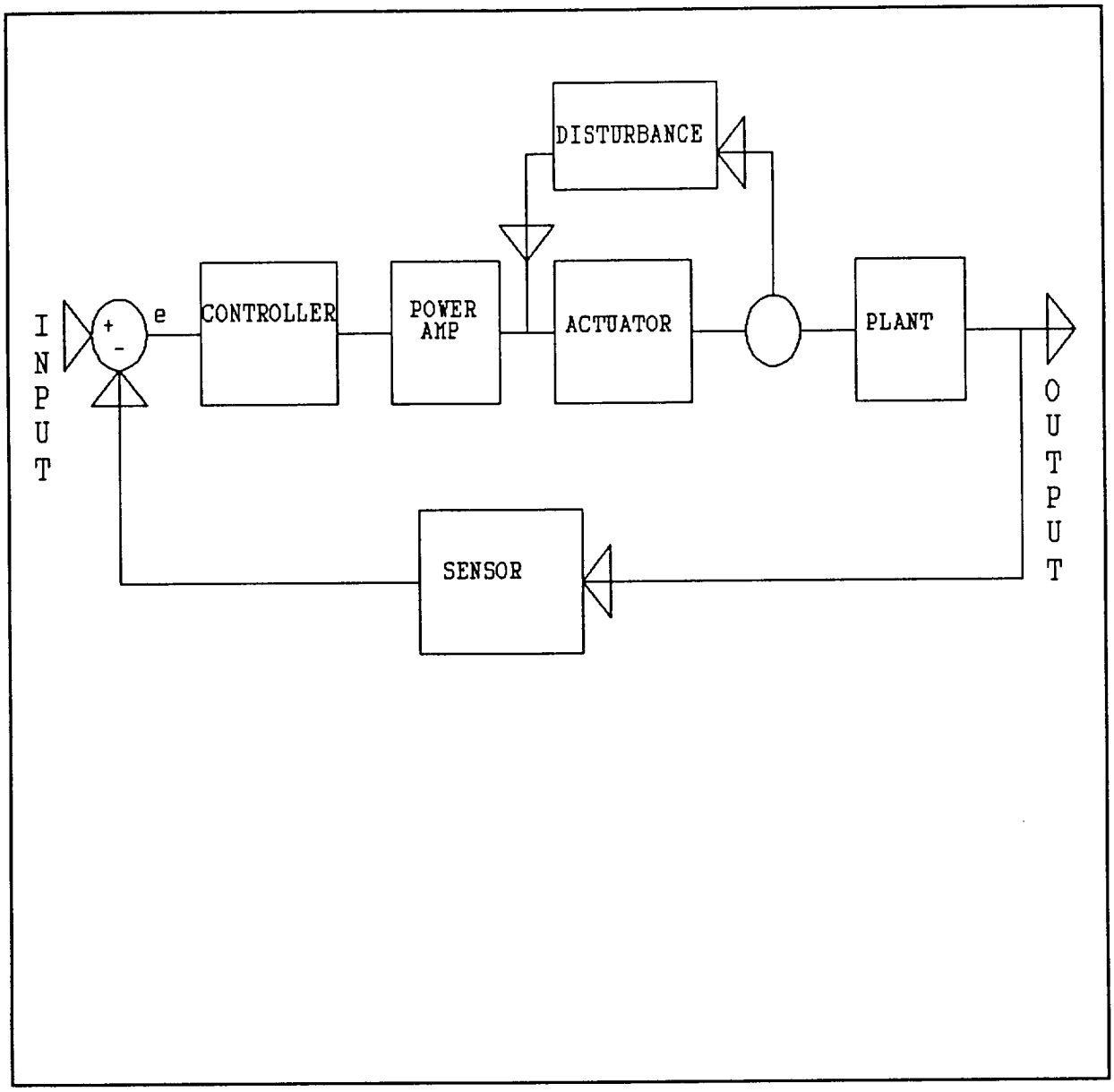
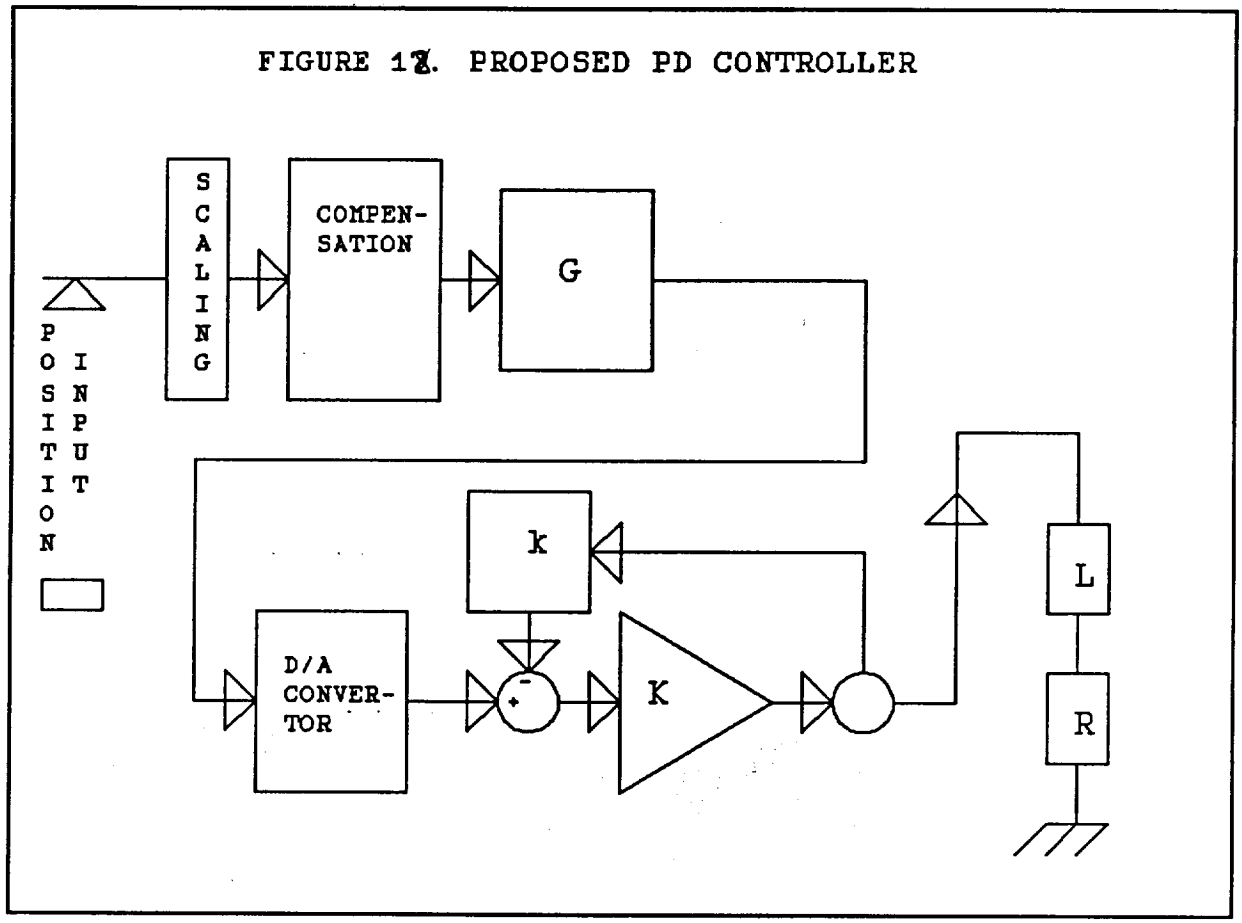


Figure 18. PD Control USED



4. PD CONTROLLER DESIGN

The controller developed for the MBA is a proportional derivative controller which is modeled entirely as a computer program. Figure 18, is a block diagram of the single loop system which consists of the above controller, plant, and position sensor feedback loop. The controller is the C program described earlier in this report. Physically, the plant consists of the KEPCO programmable power supply, the actual bearing station and a one ohm resistor on the return line to the power supply. The position sensor feedback loop consists of the position sensors, and position sensor amplifiers (see figure 9, page 23).

The transfer function, G(s) models the entire system as described above. In the development of this model some assumptions were made to simplify the end result. One assumption was that the programming time constant of the KEPCO power supply was fast enough that any roots or poles that it generated would be very far to the left on the left half plane, and hence would not effect the stability of the system. Another assumption was that the reaction times of the DT2811 patch panel and the position sensor amplifiers were also fast enough not to be included in the theoretical model. Overall, these assumptions allowed the position sensor feedback to be modeled as a constant (403.27 volts per meter for bearing station A) and the voltage to current conversion carried out by the KEPCO programmable power supply to be represented as a gain of two.

Once the theoretical model was complete it was programmed into MATLAB for analysis. Using trail and error, different values of kp

and kd were analyzed by MATLAB using a ratio of kp/kd that generated a root loci plot that was entirely on the LHP was determined. The ratio of kp/kd = 100 was determined as an optimum gain ratio and produced the root loci plot as shown. The above gain ratio resulted in stable magnetic levitation of bearing station A. In conclusion, the theoretical model is correct because it predicts stability and this is evidenced by stable operation of the MBA for the selected kp/kd ratio.

Usually K_p/K_d.

Derivation of Control Logic

VII. CONCLUSION

The third order transfer function derived in this project represents a stable control system with a gain of 100. This was used to produce a control program in the C language that, when integrated with the system electronics, operates a single axial bearing station. Thus, the objectives of this semester project have been met and the ASPS program is ready to build on these accomplishments. Operation of the entire ASPS should now be a primary goal. While there is still much work to be completed, the operation of a single station has proven that it is feasible, and has identified the aspects that must be modified to obtain such a goal.

usually not numbered

VIII. REFERENCES

1) The Development of the ASPS Vernier System, Final Report, Sperry Corporation.

2) ASPS-Final Design Review, Sperry Corporation.

3) Van de Vegte, John. Feedback Control Systems, 2nd ed., New Jersey: Prentice Hall, 1990.

4) Kernighan, Brian W. and Dennis M. Ritchie. The C Programming Language, 2nd ed. New Jersey: Prentice Hall, 1988.

5) Microsoft Corp., C For Yourself, 1990.

6) Ohanian, Hans C. Physics, New York: W.W. Norton & Co., 1985.

7) Data Translation Manuals for DT2811 & DT2819.

Appendix

```
/*************************************************************************/
/*                        ASPS                                           */
/*              Single Magnetic Bearing Assembly Code                    */
/*        MEM Senior Project Members: WE Smith, W Thomas, T Quach        */
/*               Old Dominion University - Fall 1993                     */
/* ----------------------------------------------------------------------- */
/*    Programming Language: C                    Programmer: T Quach      */
/*************************************************************************/


/*************************************************************************/
#include <graph.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <signal.h>
#include <stdlib.h>
#include <malloc.h>


/*  ASCI definitions for the function keys assoc. with the oscilloscope  */
#define UP 72
#define DOWN 80
#define LEFT 75
#define RIGHT 77
#define ENTER 28
#define ESC 27
#define F1 59
#define F2 60
#define F4 62
#define F5 63
#define F6 64
#define F9 67
#define F10 68
#define pi 3.14159
#define On 1
#define Off 0
/*************************************************************************/
   char    choice,          /* Bolean for loop termination  */
        ch,ch1,ch2,
        prnchoice,
        chin,
        plotmode,loop;
   short  Vf,Vi,           /* analoge data of sensor signal */
       Kp,Kd,           /* Gain constants */
       xvalue,analog_dat;
   int    chanin,chanout, /* I/O Channel for DAC board */
```

```
        finish,          /* Bolean for main loop termination */
        maxx,maxy,       /* Screen dimension */
        counter,ii,i,l,   /* interation loop counter */
        increment,
        temp,
        dim,x,y,        /* array counter for data */
        chd,            /* Bolean for loop termination */
        trig,            /* plot type determination */
        zoom=1,
        positionsign,
        gainupdown,
        /* constants */
        ix=25;
    double a11,a,b,c,d,e,f;
    float intergral,xold,speed;
    long backc;
/*************************************************************************/
/************************  Function definitions  ************************/
    void scrn_setup(void);
    void prn_info(void);
    void oscilloscope(void);
    void plotdat(void);
    void keyboard(void);
    void outdat(void);
    void gainplot(void);
    void positionplot(void);
    void trigplot(void);
    void compensator(void);
    void timersetup(void);
    void savedat(void);
    double getdat(double *Kp, double *Kd,short *chanin, short *chanout);


/*************************************************************************/
/************************  File definitions  ************************/
    FILE          SBAIN,     /* pointer for "constin" file */
           SBAVOLT,   /* pointer for "volt" file */
           SBAY,      /* pointer for "gap" file */
           SBAI,      /* pointer for "current" file */
           SBAFORC;   /* pointer for "force" file */

/*************************** MAIN PROGRAM  ******************************/
/*=====================  BEGIN MAIN ====================================*/

main()
{
     prn_info();
     scrn_setup();
     printf("This program can perform the following: \n");
     printf("    1) Passive Mode - no screen display just controls \n");
     printf("    2) Active Mode  - oscilloscope display with controls \n");
     printf("    3) Quit \n");
```

```
 do {
    printf("Please enter the number of your choice: ");
    choice=getchar();
    counter=0;
    Kp=0;
    Kd=0;
    chanin=0;
    chanout=0;
    switch (choice) {
     case 'P': case 'p':          /* Passive */
          printf("1");
          counter = counter + 1;
          dat_retreval();
          pd_control();
          prn_data();
          break;
     case 'A': case 'a':          /* Active */
          printf("2");
          counter = counter + 1;
          dat_retreval();
          pd_ctrlplot();
          prn_data();
          break;
     case 'Q':case 'q':           /*  Quit */
          finish = 1;
          printf("bye!");
          break;
     default:
          printf("invalid choice selected, select again\n");
          finish = 0;
          break;
     }
    } while (!finish);
    exit(1);
}
/***************---- end of main ---------- ********************************/
/*****************************************************************************/

/*****************************   SUBROUTINES  ****************************/
/*****************************************************************************/
void prn_info(void)
{
scrn_setup();
printf("                              ASPS    \n                                ");
printf("              Single Magnetic Bearing Assembly Code   \n              ");
printf("        MEM Senior Project Members: WE Smith, W Thomas, T Quach \n ");
printf("                 Old Dominion University - Fall 1993   \n              ");
printf("  \n ");
printf(" Utilizes:  1) Data Translation Dt2811 DAC board  (V 1.02\n");
printf("       2)  "       "          Pacer Timer board (V 1.00) \n");
printf("             3)  Microsoft (quick) C   (V 1.20)                \n");
printf(" Hit any key to continue");
```

```
getch();
   earscreen(_GCLEARSCREEN);
}
/*************************************************************************/
void scrn_setup(void)
{
  graphic_config();
  _setcolor(4);
  _rectangle(_GFILLINTERIOR,550,420,637,447);
  _settextposition(5,10);
}
/*************************************************************************/
double getdat(double *Kp, double *Kd,
          short *chanin, short *chanout)
{
scrn_setup();
SBAIN=fopen("c:\data\const(counter)": "w+");
printf("\n Please enter the proportional gain: ");
scanf("%d\n",&Kp);
fprintf(SBAIN,"Kp :  %d",&Kp);
printf("Please enter the derivative gain: ");
scanf("%d\n",&Kd);
fprintf(SBAIN,"Kp :  %d",&Kp);
printf("Please enter the signal input channel: ");
scanf("%d\n",&chanin);
fprintf(SBAIN,"chan. in :  %s",&chanin);
printf("Please enter the signal output channel: ");
scanf("%d\n",&chanout);
fprintf(SBAIN,"chan. out :  %d",&chanout);
fclose(SBAIN);
)

/*************************************************************************/
double pd_control(double *Kp, double *Kd,
           double *chanin, double *chanout)
{
     ch=ESC;
     while(ch1!=ESC)
     {
     _setcolor(4)
     _rectangle (_GFILLINTERIOR,550,420,637,447);
     _settextposition(28,71);
     printf("RUNNING");
     _settextposition(29,71);
     printf("ESC = QUIT");
     timer();
     _settextposition(12,24);
     while(ch1!=ESC)
          {
 /*            SBAVOLT=fopen("volt","w+");      */

          keyboard();
```

```
    Vi=LPAD(chanin,analog_dat);
    while(ch2!=ESC)
    }

        xold=xvalue;
        LPAV(chanin,gain,&xvalue);
        Vf=xvalue;
        printf("%i\n",xvalue);
        xvalue=xvalue-2047;
        intergral=intergral+.01*xvalue;

        speed=(xvalue-xold) /.003;
        analog_dat=-.5*xvalue-.005*speed-0*intergral;
        Vc=analog_dat;
        analog_dat=analog_dat+2314;
        LPDV(chanout,analog_dat);

        save_dat();
        fprintf(SBAVOLT,"%h",&Vi);
        Vi=Vf;
    }
    analog_dat=2047;
    LPDV(0,analog_dat);

}
/*****************************************************************************/
uble pd_ctrlplot(double *Kp, double *Kd,
          double *chanin, double *chanout)
{
    ch=ESC;
    while(ch1!=ESC)
    {
    _setcolor(4)
    _rectangle (_GFILLINTERIOR,550,420,637,447);
    _settextposition(28,71);
    printf("RUNNING");
    _settextposition(29,71);
    printf("ESC = QUIT");
    timer();
    _settextposition(12,24);
    while(ch1!=ESC)
        {
        ofp=fopen("volt","w+");  */

        keyboard();
        start_time();
        _setvideomode(_VERS16COLOR);
        oscilloscope();
        Vi=LPAD(chanin,analog_dat);
        while(ch2!=ESC)
        }
            LPDV(chanout,analog_dat);
```

```
            Vf=LPAD(chanin,analog_dat);
            samplerate=prn_time();
            Vc=(Kp*Vi)+Kd*((Vf-Vi)/samplerate);
            LPCV(&Vc);
            sav_dat();
           /*    fprintf(ofp,Vi);  */
            Vi=Vf;
      if(chd!=0)
          {
        if(!trig || ix<474)
          {
            if(plotmode=='a') plotdat();
            if(plotmode=='s')
              {
             switch(ch2)
                {
                   case 'x': if(j==0) trigplot();
                   break;
                   case 'y': if(j==1) trigplot();
                   break;
                   case 'z': if(j==2) trigplot();
                   break;
                   case 'm': if(j==3) trigplot();
                   break;
                   case 'n': if(j==4) trigplot();
                   break;
                }
              }
          }
          }
    }

/*****************************************************************/
     if(trig==0)
     if(ix>=475) ix=25;/* reset screen position to begining when at end */
     if(ix<=475) ix++;
         /* screen horizontal advance  chk plotdat     */

   }
   _setcolor(2);
   _rectangle(_GFILLINTERIOR,550,420,637,477);
   _settextposition(28,71);
   _outtext("R-Run");
   _settextposition(29,71);
   _outtext("ESC-exit");
   ch='.';
   ch1=getch();
 }
 }
 }
/*****************************************************************/
/* saves data to file 'volt'   */
```

```
void savedat(void)

       int i;
       FILE *SBAVOLT;


      if((SBAVOLT=fopen(volt,"w+"))!=NULL)
        {
          ch=ESC;
          while(ch1!=ESC)
              {
                fprintf(SBAVOLT,"%s",&V);
                fprintf(SBAVOLT,"\n");
              }
          fclose(fp);
        }
}
/*************************************************************************/
/*  loads data from file 'volt'    */

void getdat(void)
{
     int i,j,k,dim;

     if((SBAVOLT=fopen(volt,"r+"))!=NULL)
     {
      printf("Data loading \n");
      fscanf(fp,"%d",&dim);
      printf(" Number of data points are: %d \n ",dim);
      for(k=0;k<=dim-1;k++)
      {
         fscanf(fp,"%d",&V[k]);
      }
      printf(" Data was successfully loaded, hit Enter");
      fclose(fp);
     }
       else
       fprintf(stderr,"Oops file error");

}
/*************************************************************************/
/*   Calculates the force relationships for printing  */
double calc_force(double *)
{
printf("the force calculation");
(fp=fopen(volt,"r+")
printf("Data loading \n");
      fscanf(fp,"%d",&dim);
      printf(" Number of data points are: %d \n ",dim);
      for(increment=0;increment<=dim-1;increment++)
      {
         fscanf(fp,"%d",&V[increment]);
```

```
        F[incrememt]=(  /permiability);
      }
      fclose(fp);


return F;
}
/*************************************************************************/
/*   Calculates the position relationships for printing  */
double calc_y(double *)
{
printf("the position calculation");
(fp=fopen(volt,"r+")
printf("Data loading \n");
      fscanf(fp,"%d",&dim);
      printf(" Number of data points are: %d \n ",dim);
      for(increment=0;increment<=dim-1;increment++)
      {
         fscanf(fp,"%d",&V[increment]);
         y[incrememt]=(  /permiability);
      }
      fclose(fp);
return y;
}
/*************************************************************************/
/*   Calculates the current relationships for printing  */
  uble calc_current(double *)
{
printf("the curent calculation");
(fp=fopen(volt,"r+")
printf("Data loading \n");
      fscanf(fp,"%d",&dim);
      printf(" Number of data points are: %d \n ",dim);
      for(increment=0;increment<=dim-1;increment++)
      {
         fscanf(fp,"%d",&V[increment]);
         current[incrememt]=(  /permiability);
      }
      fclose(fp);
return current;
}
/*************************************************************************/

/*************************************************************************/
double prn_data(double * )
{
printf("Would you like a hardcopy of the results (Y/N): " );
prnchoice=getch();
switch (prnchoice) {
    case 'Y':case 'y':
     printf("What type of data: Force, Position, Current, Voltage, All");
     printf("Please enter the FIRST letter of choice:");
```

```
    datachoice=getch();
    switch(datachoice){
         case 'F': case 'f':
         calc_force();
         prn_format();
         break;
       . case 'P': case'p':
         calc_y();
         prn_format();
         break;
         case 'C': case 'c':
         calc_current();
         prn_format();
         break;
         case 'V': case 'v':
         fp=fopen(volt,"r++");
         fscanf(fp,"%d",&V);
         prn_format();
         break;
         case 'A': case 'a':
         calc_force();
         calc_y();
         calc_current();
         fp=fopen(volt,"r++");
         fscanf(fp,"%d",&V);
         prn_format();
         break;
    }
   break;
   case 'N':case 'n':
    printf("No printout selected");
    break;
   }
}
/*************************************************************************/
double prn_format(double *)
{
printf("prn format exec");
}
/*************************************************************************/
/* keyboard: User interaction through keyboard                           */
void keyboard(void)
{
   char chin;
   if(kbhit()!=0)
     {
      chin=getch();
/*    if(!chin)     chin=getch(); */
      switch(chin)
     {
        case 'X':case 'x': ch2='x';
        .       _setcolor(8);
```

```
            _rectangle(_GFILLINTERIOR,35,320,39,478);
            _setcolor(12);
            _rectangle(_GFILLINTERIOR,35,320,39,340);
            if(plotmode=='s')
            {
               _setcolor(plotcolor);
               _rectangle(_GFILLINTERIOR,20,5,480,290);
            }
            break;
 case 'Y':case 'y': ch2='y';
            _setcolor(8);
            _rectangle(_GFILLINTERIOR,35,320,39,478);
            _setcolor(12);
            _rectangle(_GFILLINTERIOR,35,352,39,372);
            if(plotmode=='s')
            {
               _setcolor(plotcolor);
               _rectangle(_GFILLINTERIOR,20,5,480,290);
            }
            break;
 case 'Z':case 'z': ch2='z';
            _setcolor(8);
            _rectangle(_GFILLINTERIOR,35,320,39,478);
            _setcolor(12);
            _rectangle(_GFILLINTERIOR,35,384,39,404);
            if(plotmode=='s')
            {
               _setcolor(plotcolor);
               _rectangle(_GFILLINTERIOR,20,5,480,290);
            }
            break;
 case 'M':case 'm': ch2='m';
            _setcolor(8);
            _rectangle(_GFILLINTERIOR,35,320,39,478);
            _setcolor(12);
            _rectangle(_GFILLINTERIOR,35,416,39,436);
            if(plotmode=='s')
            {
               _setcolor(plotcolor);
               _rectangle(_GFILLINTERIOR,20,5,480,290);
            }
            break;
 case 'N':case 'n': ch2='n';
            _setcolor(8);
            _rectangle(_GFILLINTERIOR,35,320,39,478);
            _setcolor(12);
            _rectangle(_GFILLINTERIOR,35,448,39,468);
            if(plotmode=='s')
            {
               _setcolor(plotcolor);
               _rectangle(_GFILLINTERIOR,20,5,480,290);
            }
```

```
          break;
case 'S': case 's': plotmode='s';
          _setcolor(plotcolor);
          _rectangle(_GFILLINTERIOR,20,5,480,290);
          break;
case 'A': case 'a': plotmode='a';
          break;
case 'T': case 't':   trig=1;
          break;
case 'C': case 'c': trig=0;
          break;
case 0:
   chin=getch();
   switch(chin)
   {
   case RIGHT: gainupdown=1;
          gainplot();
          break;
   case LEFT: gainupdown=-1;
          gainplot();
          break;
   case UP: positionsign=1;
          positionplot();
          break;
   case DOWN: positionsign=-1;
          positionplot();
          break;
   case F1: if(zoom>1) zoom=zoom-1;
          _settextposition(1,60);
          printf("%i ",zoom);
          break;
   case F2: if(zoom<10) zoom=zoom+1;
          _settextposition(1,60);
          printf("%i ",zoom);
          break;
   case F4: savedat();        /* Save step responses if needed */
          break;
   case F5:
       printf("no operation function for F5");
          break;
   case F6:
       printf("no operation function for F6");;
          break;
   case F9: rota=0;
          _settextcolor(8);
          _settextposition(19,75);
          _outtext("on");
          _settextcolor(4);
          _settextposition(18,75);
          _outtext("off");

          break;
```

```
          case F10: rota=1;
                _settextcolor(8);
                _settextposition(18,75);
                _outtext("off");
                _settextcolor(4);
                _settextposition(19,75);
                _outtext("on");
                break;
         }
         break;
         case 'E':case 'e':  chd=1;
                break;
         case 'D':case 'd': chd=0;
                break;
         case ESC: ch=ESC; break;
        }
        }
       chin='.';
}

/*****************************************************************************/
/* oscilloscope: signal display screen setup       */
void oscilloscope(void)
{
   int ll,jj,gainstart,posibar;
/* initialization of the screen clear array */
    for(ll=0;ll<=5;++ll)
     {
     for(jj=0;jj<=500;++jj)
          olddat[jj][ll]=20;
      }
/* display of the oscilloscope     screen          */
     _clearscreen(_GCLEARSCREEN);
     _setcolor(13);
     _rectangle(_GBORDER,0,0,639,479);
     _setcolor(plotcolor);
     _rectangle(_GFILLINTERIOR,20,5,480,290);
     _settextposition(4,2);
     printf("X");
     _settextposition(7,2);
     printf("Y");
     _settextposition(10,2);
     printf("Z");
     _settextposition(13,2);
     printf("M");
     _settextposition(16,2);
     printf("N");
     _setcolor(7);
     for(ll=52;ll<=244;ll=ll+48)
     {
        _moveto(19,ll);
        _lineto(24,ll);
```

```
        _moveto(476,11);
        _lineto(484,11);
    }
/*****************************************************/
/* Gain control knobs Display          */
    _setcolor(12);
    _rectangle(_GBORDER,2,300,480,478);
    _settextposition(20,2);
    _settextcolor(14);
    _outtext("Feedback gain control");
    _settextposition(20,40);
    _outtext("Position Control");
    _setcolor(5);
    jj=0;
    for(11=21;11<=29;11=11+2)       /* lableing the gain knobs */
      {
         _settextposition(11,27);
         printf("%4.3f",gain[jj]);
         jj++;
         _settextposition(11,4);
         printf("0");
       }
    for(11=319;11<=447;11=11+32)      /* gain control bar display */
         _rectangle(_GBORDER,42,11,196,11+22);
    jj=0;
    for(11=320;11<=448;11=11+32)       /* Presetting gain display */
         {                    /* on the screen          */
         gainstart=(int)(gain[jj]*80)+40;
         _setcolor(6);
         _rectangle(_GFILLINTERIOR,40,11,gainstart+3,11+20);
         ++jj;
    }
/* display of position control knobs  */

    _setcolor(4);
     for(11=320;11<=448;11=11+32)       /* position control bar display */
         _rectangle(_GBORDER,310,11,460,11+20);
    _setcolor(10);
    jj=0;
    for(11=320;11<=448;11=11+32)

      {  posibar=385+(int)(position[jj]*75);
         _rectangle(_GFILLINTERIOR,posibar,11,posibar+1,11+20);
         ++jj;
    }
/*****************************************/
/* labling of gain knob display    */
    _settextposition(21,2);
    _outtext("X");            /* x or axial */
    _settextposition(23,2);
    _outtext("Y");            /* y or side  */
    _settextposition(25,2);
```

```
    _outtext("Z");            /* z or vertical */
    _settextposition(27,2);
    _outtext("M");            /* M - pitch movement */
    _settextposition(29,2);
    _outtext("N");          /*    N - yaw movement */
/************************************/
/* Display of options */
    _settextposition(2,62);
    printf("E- enable plot");
    _settextposition(3,62);
    printf("D- disable plot");
    _settextposition(5,62);
    printf("Select Channel");
    _settextposition(6,62);
    printf("   Press  ");
    _settextposition(7,62);
    printf(" X Y Z M N");

    _settextposition(9,62);
    printf("S- single plot") ;
    _settextposition(10,62);
    printf("A - all plots");
    _settextposition(11,62);
    printf("T - triggered ");
    _settextposition(12,62);
    printf("C - continuous");
    _settextposition(13,62);
    printf("F1- unzoom plot");
    _settextposition(14,62);
    printf("F2- zoom plot");
    _settextposition(15,62);
    printf("F4- Save data");
    _settextposition(16,62);
    printf("F5- - Yaw ");
    _settextposition(17,62);
    printf("F6- + Yaw ");
    _settextposition(18,62);
    printf("F9-  AutoYaw off");
     _settextcolor(4);
     _settextposition(18,75);
     _outtext("off");

    _settextposition(19,62);
    printf("F10- AutoYaw on");

    _settextposition(24,62);
    printf("Yaw Error:");
    _settextposition(22,62);
    printf("Yaw Angle:");


    }
```

```
/*****************************************************************/
   Plotdat: plot of signals */
void plotdat(void)
{
        •   kv=4+(j+1)*48-(int)(v*2*zoom);
          if(kv>=290) kv=290;
          _setcolor(plotcolor);
          _setpixel(ix,olddat[ix][j+1]);
          _setcolor(datcolor);
          _setpixel(ix,kv);
          olddat[ix][j+1]=kv;
 }
/*****************************************************************/
/* trigplot: plot of signals */
void trigplot(void)
{    int plotcenter=148;

           kv=plotcenter-(int)(v*10*zoom);
          if(kv>=290) kv=290;
          _setcolor(plotcolor);
          _setpixel(ix,olddat[ix][j+1]);
          _setcolor(datcolor);
          _setpixel(ix,kv);
          olddat[ix][j+1]=kv;
/*              tmp[j]=kv;                    */
 }
/*****************************************************************/
/* gainplot: to display gain on each degree of freedom graphically */
void gainplot(void)
{
     int gainknob,knobcolor;
     switch(ch2)
     {
        case 'x':
           if((gain[0]>=0) && (gain[0]<=2))
            {
          knobcolor=3*(gainupdown+1);
          _setcolor(knobcolor);
          gainknob=40+(int)(gain[0]*80.0);
          _rectangle(_GFILLINTERIOR,gainknob,320,gainknob+3,340);
          gain[0]=gain[0]+gainupdown*gainstep;
          _settextposition(21,27);
          printf("%4.3f",gain[0]);

            }
           else
            if(gain[0]<0) gain[0]=0;
            if(gain[0]>2) gain[0]=2;
          break;
        case 'y':
           if((gain[1]>=0) && (gain[1]<=2))
          {
```

```
   _setcolor(knobcolor);
   gainknob=40+(int)(gain[1]*80.0);
   _rectangle(_GFILLINTERIOR,gainknob,352,gainknob+3,372);
   gain[1]=gain[1]+gainupdown*gainstep;
   _settextposition(23,27);
   printf("%4.3f",gain[1]);
     }
    else
     if(gain[1]<0) gain[1]=0;
     if(gain[1]>2) gain[1]=2;
 break;
case 'z':
    if((gain[2]>=0) && (gain[2]<=2))
   {
   knobcolor=3*(gainupdown+1);
   _setcolor(knobcolor);
    gainknob=40+(int)(gain[2]*80.0);
    _rectangle(_GFILLINTERIOR,gainknob,384,gainknob+3,404);
    gain[2]=gain[2]+gainupdown*gainstep;
    _settextposition(25,27);
    printf("%4.3f",gain[2]);
   }
    else
     if(gain[2]<0) gain[2]=0;
     if(gain[2]>2) gain[2]=2;
 break;
case 'm':
    if((gain[3]>=0) && (gain[3]<=2))
    {
   knobcolor=3*(gainupdown+1);
   _setcolor(knobcolor);
   gainknob=40+(int)(gain[3]*80.0);
   _rectangle(_GFILLINTERIOR,gainknob,416,gainknob+3,436);
   gain[3]=gain[3]+gainupdown*gainstep;
   _settextposition(27,27);
   printf("%4.3f",gain[3]);
    }
    else
     if(gain[3]<0) gain[3]=0;
     if(gain[3]>2) gain[3]=2;
 break;
case 'n':
        if((gain[4]>=0) && (gain[4]<=2))
    {
   knobcolor=3*(gainupdown+1);
   _setcolor(knobcolor);
   gainknob=40+(int)(gain[4]*80.0);
   _rectangle(_GFILLINTERIOR,gainknob,448,gainknob+3,468);
   gain[4]=gain[4]+gainupdown*gainstep;
   _settextposition(29,27);
   printf("%4.3f",gain[4]);
    }
```

```
        else
        if(gain[4]<0) gain[4]=0;
        if(gain[4]>2) gain[4]=2;
                    break;


    }
}

/*************************************************************************/
/* positionplot: control and display of position knobs                   */
void positionplot(void)
{
      int knobcolor,positionknob,bkcolor;
      knobcolor=10;
      bkcolor=4;
    if(trig==1)
       ix=25;
       sampler=0;  /* counter for collecting data in ISR */

    switch(ch2)
    {

       case 'x':
         _setcolor(0);
         _rectangle(_GFILLINTERIOR,310,320,461,340);
         _setcolor(bkcolor);
         _rectangle(_GBORDER,310,320,461,340);
         _setcolor(knobcolor);
         position[0]=position[0]+positionsign*positionstep;
         if(position[0]<-1) position[0]=-1;
         if(position[0]>1) position[0]=1;
         positionknob=385+(int)(position[0]*75.0);
         _rectangle(_GFILLINTERIOR,positionknob,320,positionknob+1,340);
         break;
       case 'y':
         _setcolor(0);
         _rectangle(_GFILLINTERIOR,310,352,461,372);
         _setcolor(bkcolor);
         _rectangle(_GBORDER,310,352,461,372);
         _setcolor(knobcolor);
         position[1]=position[1]+positionsign*positionstep;
         if(position[1]<-1) position[1]=-1;
         if(position[1]>1) position[1]=1;
         positionknob=385+(int)(position[1]*75.0);
         _rectangle(_GFILLINTERIOR,positionknob,352,positionknob+1,372);
         break;
       case 'z':
         _setcolor(0);
         _rectangle(_GFILLINTERIOR,310,384,461,404);
         _setcolor(bkcolor);
         _rectangle(_GBORDER,310,384,461,404);
```

```
        _setcolor(knobcolor);
        position[2]=position[2]+positionsign*positionstep;
        if(position[2]<-1) position[2]=-1;
        if(position[2]>1) position[2]=1;
        positionknob=385+(int)(position[2]*75.0);
          _rectangle(_GFILLINTERIOR,positionknob,384,positionknob+1,404);
       break;
      case 'm':
        _setcolor(0);
        _rectangle(_GFILLINTERIOR,310,416,461,436);
        _setcolor(bkcolor);
        _rectangle(_GBORDER,310,416,461,436);
        _setcolor(knobcolor);
        position[3]=position[3]+positionsign*positionstep;
        if(position[3]<-1) position[3]=-1;
        if(position[3]>1) position[3]=1;
        positionknob=385+(int)(position[3]*75.0);
          _rectangle(_GFILLINTERIOR,positionknob,416,positionknob+1,436);
       break;
      case 'n':
        _setcolor(0);
        _rectangle(_GFILLINTERIOR,310,448,461,468);
        _setcolor(bkcolor);
        _rectangle(_GBORDER,310,448,461,468);
        _setcolor(knobcolor);
        position[4]=position[4]+positionsign*positionstep;
        if(position[4]<-1) position[4]=-1;
        if(position[4]>1) position[4]=1;
        positionknob=385+(int)(position[4]*75.0);
        _rectangle(_GFILLINTERIOR,positionknob,448,positionknob+1,468);
         break;


    }
}
/*************************************************************************/
/*-----------------------------------------------------------------------*/
```

```
/*****************************************************************/
/*                          L.A.M.S.T.F.                         */
/*          (Large Angle Magnetic Suspention Test Fixture)       */
/* Controller program for the five degrees of freedom magnetic   */
   suspension system, at NASA Langley Hampton, Virginia.         */
/*                   Version 2 programmed by :                   */
/*                        Lucas Foster                           */
/*                   Old Dominion University                     */
/*               Based on Version One Programmed By:             */
/*                       Mehran Ghofrani                         */
/*                   Old Dominion University                     */
/*                                                               */
/*****************************************************************/
#include <graph.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <float.h>
#include <dos.h>
#include <bios.h>
#include <signal.h>
#include <stdlib.h>
#include <malloc.h>


#define Address 512          /* A/D base address    */
#define ADC_2   0x210        /* Timer board Address */
#define DDAbase 816          /* D/A base address    */
#define UP 72
#define DOWN 80
#define LEFT 75
#define RIGHT 77
#define ENTER 28
#define ESC 27
#define F1 59
#define F2 60
#define F4 62
#define F5 63
#define F6 64
#define F9 67
#define F10 68
#define pi 3.14159
#define On 1
#define Off 0
/*****************************************************************/
#pragma intrinsic(outp,inp)

/* External Mode call (Function ) Prototype */
extern mscl_das40(int*,int*,int*);

#define Inport  512

/* das-40 variables */
int mode,flag,params[10];

/* Pointers to Allocated ram Buf's    */
unsigned int *DMA_buf,*bufoffset;

unsigned int chan=10;
```

```
/* Channel and Gain Array to use by Mode 2 */
int chans[5]={0,1,2,3,4};
int gains[5]={0,0,0,0,0};

: far *chans_ptr;
int far *gains_ptr;


/*****************************************************************/
  unsigned int adcsr;
int dec,sampler;
    int y11,x1,samplrate,diviser;

    int intnum=0x0d;       /* interrupt vector number */
    int input,k,j,num,plotcolor=0,datcolor=15;
    int ii,i,l,x,y,chd,olddat[500][5];
    int zoom=1,ang=0;
    int positionsign,gainupdown,trig;
    int dec,tmp[5]={53,101,149,197,245},kv,ix=25;
    double a11,a,b,c,d,e,f,current[5],savdat[5][500];
    double v,v1[25],v2[25],vsensor[5],initialoffset[5]={4.00,1.631,-4.133,-4.133,
    double offset[5]={4.86,1.631,-4.133,-4.133,1.631};
    double gain[5]={.5,.5,.5,.1,.08},gainstep=.001,position[5],positionstep=.1;
    double mix[5][5]={{1.0,0.0,-1.0,0.625,0.0},
                      {-0.809,0.618,-0.309,1.0,1.0},
                      { 0.309,-1.0,0.809,0.768,0.618},
                      { 0.309,1.0,0.809,0.768,-0.618},
                      {-0.809,-0.618,-0.309,1.0,-1.0}};

    double mixall[5][5][61],yawangle=0,yawanglestep=0.5,filt[5],autoyaw=0;
    double cosdat[800];
    int maxnum,yawerrorlimit;
    char ch,ch1,ch2,plotmode,loop;
    int rota=0;
    long backc;
/*****************************************************************/

    void (_interrupt _far *oldnum)(void);
    void _interrupt _far datin(void);
/*****************************************************************/

    void process_error(void);
    void initialize(void);
    void oscilloscope(void);
    void timerset(void);
    void adcinitialize(void);
    void plotdat(void);
    void keyboard(void);
    void outdat(void);
    void gainplot(void);
    void positionplot(void);
    void trigplot(void);
    void compensator(void);
    void timersetup(void);
    void acknowledge(void);
    void savedat(void);
    void getdat(void);
    void newmixer(void);
    void cosmak(void);
    void shutdown(void);
```

```
main()
{
/*************************************************************/
/* ISR vector setup */
/*************************************************************/
    oldnum=_dos_getvect(intnum);          /* 0x0d IRQ-5 */
    _disable();                           /* Disable all interrupts */
    _dos_setvect(intnum,datin);   /* set new interrupt vector table */
    _enable();
   outp(0x21,0x00);                    /* send acknowledgement signal */
   printf("here1");
   outp(0x20,0x20);                    /* to the interrrupt handler */
   printf("here");

    _setvideomode(_VRES16COLOR);  /*select video configuration */

/*************************************************************/
     cosmak();            /* calculate cosine values for every .5 deg */
     getdat();             /* Get mixer matrix data */
     adcinitialize();             /* Initialize the A/D converter board */
     printf(" Enter Sample rate Please :  ");
     scanf("%d",&samplrate);      /* Get the sampling rate */
     initialize();                /* Compute The compensators parameters */
     oscilloscope();              /* Display the Controller Screen */
     timersetup();                /* Set timer to interrupt at sample-rate */
/*************************************************************/
/* Controller loop */


      ch=ESC;
      while(ch1!=ESC)
      {
        _setcolor(4);
        _rectangle(_GFILLINTERIOR,550,420,637,477);
        _settextposition(28,71);
        printf("RUNNING");
        _settextposition(29,71);
        printf("ESC- quit");
        acknowledge();

   while(ch!=ESC)
        {

        keyboard();            /* Check for user input */

/*      _settextposition(12,24);
        printf(" %8.5f",v1[14]); */


     filt[4]=filt[3];
     filt[3]=filt[2];                    /*  YAW error values for */
     filt[2]=filt[1];                    /*  several samples  */
     filt[1]=filt[0];
     filt[0]=v1[14];
     autoyaw=.5*(filt[0]+filt[1]);       /* filter yaw error signal */
     _settextposition(24,72);
     printf("%4.3f",autoyaw);            /* display the yaw error */
```

```
if(rota==1)                          /* Check for automatic yaw sensing */
 {                                   /* option  */

          if(autoyaw>=2)
            {
                  /* perform two yaw correction steps */

                  yawangle=yawangle+yawanglestep;
                  newmixer();
                  yawangle=yawangle+yawanglestep;
                  newmixer();
             }
          if(autoyaw<=-2)
             {
                  yawangle=yawangle-yawanglestep;
                  newmixer();
                  yawangle=yawangle-yawanglestep;
                  newmixer();
             }
 }
  for(j=0;j<=4;++j)
   {
       v=v1[2+3*j];

       if(sampler<=2)
           {                          ← Yellow
             _setcolor(14);
             _rectangle(_GFILLINTERIOR,600,330,620,325);
           }
       else
         if(sampler>200)
           {                          ← Blue
             _setcolor(1);
             _rectangle(_GFILLINTERIOR,600,330,620,325);
           }

      if(chd!=0)
         {
           if(!trig || ix<474)
             {
               if(plotmode=='a') plotdat();
               if(plotmode=='s')
                 {
                   switch(ch2)
                      {
                         case 'x': if(j==0) trigplot();
                         break;
                         case 'y': if(j==1) trigplot();
                         break;
                         case 'z': if(j==2) trigplot();
                         break;
                         case 'm': if(j==3) trigplot();
                         break;
                         case 'n': if(j==4) trigplot();
                         break;
                      }
                 }
             }
         }
}
```

```
/*************************************************************/
      if(trig==0)
         if(ix>=475) ix=25;/* reset screen position to begining when at end */
      if(ix<=475) ix++;
                /* screen horizental advance    chk plotdat      */



 }
    _setcolor(2);
    _rectangle(_GFILLINTERIOR,550,420,637,477);
    _settextposition(28,71);
    _outtext("R-Run");
    _settextposition(29,71);
    _outtext("ESC-exit");
    ch='.';
    ch1=getch();
 }
/*************************************************************************/
    _clearscreen(_GCLEARSCREEN);
     _settextposition(1,1);
     printf("The gains are:\n");
     for(j=0;j<=4;++j)
     printf("Gain[%d]=%8.5f\n",j,gain[j]);
     _settextposition(20,30);
/*************************************************************************/
    _setvideomode(_DEFAULTMODE);
    shutdown();                    /* reset vector table etc. */

/**************---- end of main ---------- ********************************/
/*************************************************************************/
/* oscilloscope: signal display screen setup       */
void oscilloscope(void)
{
   int ll,jj,gainstart,posibar;
/* initialization of the screen clear array */
    for(ll=0;ll<=5;++ll)
     {
        for(jj=0;jj<=500;++jj)
                olddat[jj][ll]=20;
      }
/* display of the oscilloscope  screen       */
     _clearscreen(_GCLEARSCREEN);
        _setcolor(13);
        _rectangle(_GBORDER,0,0,639,479);
        _setcolor(plotcolor);
        _rectangle(_GFILLINTERIOR,20,5,480,290);
        _settextposition(4,2);
        printf("X");
        _settextposition(7,2);
        printf("Y");
        _settextposition(10,2);
        printf("Z");
        _settextposition(13,2);
        printf("M");
        _settextposition(16,2);
        printf("N");
        _setcolor(7);
        for(ll=52;ll<=244;ll=ll+48)
```

```
        {
           _moveto(19,11);
           _lineto(24,11);
           _moveto(476,11);
           _lineto(484,11);
        }
/*****************************************************/
/* Gain control knobs Display                        */
        _setcolor(12);
        _rectangle(_GBORDER,2,300,480,478);
        _settextposition(20,2);
        _settextcolor(14);
        _outtext("Feedback gain control");
        _settextposition(20,40);
        _outtext("Position Control");
        _setcolor(5);
        jj=0;
        for(11=21;11<=29;11=11+2)        /* lableing the gain knobs */
          {
             _settextposition(11,27);
             printf("%4.3f",gain[jj]);
             jj++;
             _settextposition(11,4);
             printf("0");
          }
        for(11=319;11<=447;11=11+32)       /* gain control bar display */
             _rectangle(_GBORDER,42,11,196,11+22);
        jj=0;
        for(11=320;11<=448;11=11+32)        /* Presetting gain display */
              {                             /* on the screen          */
             gainstart=(int)(gain[jj]*80)+40;
             _setcolor(6);
             _rectangle(_GFILLINTERIOR,40,11,gainstart+3,11+20);
             ++jj;
        }
/* display of position control knobs  */

        _setcolor(4);
         for(11=320;11<=448;11=11+32)      /* position control bar display */
             _rectangle(_GBORDER,310,11,460,11+20);
        _setcolor(10);
        jj=0;
        for(11=320;11<=448;11=11+32)

          {     posibar=385+(int)(position[jj]*75);
                _rectangle(_GFILLINTERIOR,posibar,11,posibar+1,11+20);
                ++jj;
        }
/*************************************/
/* labling of gain knob display    */
        _settextposition(21,2);
        _outtext("X");                  /* x or axial */
        _settextposition(23,2);
        _outtext("Y");                  /* y or side  */
        _settextposition(25,2);
        _outtext("Z");                  /* z or vertical */
        _settextposition(27,2);
        _outtext("M");                  /* M - pitch movement */
        _settextposition(29,2);
        _outtext("N");                 /*  N - yaw movement */
```

```
/************************************/
/* Display of options */
        _settextposition(2,62);
        printf("E- enable plot");
        _settextposition(3,62);
        printf("D- disable plot");
        _settextposition(5,62);
        printf("Select Channel");
        _settextposition(6,62);
        printf("   Press  ");
        _settextposition(7,62);
        printf(" X Y Z M N");

        _settextposition(9,62);
        printf("S- single plot") ;
        _settextposition(10,62);
        printf("A - all plots");
        _settextposition(11,62);
        printf("T - triggered ");
        _settextposition(12,62);
        printf("C - continuous");
        _settextposition(13,62);
        printf("F1- unzoom plot");
        _settextposition(14,62);
        printf("F2- zoom plot");
        _settextposition(15,62);
        printf("F4- Save data");
        _settextposition(16,62);
        printf("F5- - Yaw ");
        _settextposition(17,62);
        printf("F6- + Yaw ");
        _settextposition(18,62);
        printf("F9-  AutoYaw off");
         _settextcolor(4);
         _settextposition(18,75);
         _outtext("off");

        _settextposition(19,62);
        printf("F10- AutoYaw on");

        _settextposition(24,62);
        printf("Yaw Error:");
        _settextposition(22,62);
        printf("Yaw Angle:");


      }
/*********************************************************************/
/* keyboard: User interaction through keyboard                       */
void keyboard(void)
{
   char chin;
   if(kbhit()!=0)
     {
      chin=getch();
      if(!chin) chin=getch(); */
      switch(chin)
        {
           case 'X':case 'x': ch2='x';
                              _setcolor(8);
```

```
                    _rectangle(_GFILLINTERIOR,35,320,39,478);
                    _setcolor(12);
                    _rectangle(_GFILLINTERIOR,35,320,39,340);
                    if(plotmode=='s')
                    {
                       _setcolor(plotcolor);
                       _rectangle(_GFILLINTERIOR,20,5,480,290);
                    }
                    break;
 case 'Y':case 'y': ch2='y';
                    _setcolor(8);
                    _rectangle(_GFILLINTERIOR,35,320,39,478);
                    _setcolor(12);
                    _rectangle(_GFILLINTERIOR,35,352,39,372);
                    if(plotmode=='s')
                    {
                       _setcolor(plotcolor);
                       _rectangle(_GFILLINTERIOR,20,5,480,290);
                    }
                    break;
 case 'Z':case 'z': ch2='z';
                    _setcolor(8);
                    _rectangle(_GFILLINTERIOR,35,320,39,478);
                    _setcolor(12);
                    _rectangle(_GFILLINTERIOR,35,384,39,404);
                    if(plotmode=='s')
                    {
                       _setcolor(plotcolor);
                       _rectangle(_GFILLINTERIOR,20,5,480,290);
                    }
                    break;
case 'M':case 'm': ch2='m';
                    _setcolor(8);
                    _rectangle(_GFILLINTERIOR,35,320,39,478);
                    _setcolor(12);
                    _rectangle(_GFILLINTERIOR,35,416,39,436);
                    if(plotmode=='s')
                    {
                       _setcolor(plotcolor);
                       _rectangle(_GFILLINTERIOR,20,5,480,290);
                    }
                    break;
 case 'N':case 'n': ch2='n';
                    _setcolor(8);
                    _rectangle(_GFILLINTERIOR,35,320,39,478);
                    _setcolor(12);
                    _rectangle(_GFILLINTERIOR,35,448,39,468);
                    if(plotmode=='s')
                    {
                       _setcolor(plotcolor);
                       _rectangle(_GFILLINTERIOR,20,5,480,290);
                    }
                    break;
 case 'S': case 's': plotmode='s';
                    _setcolor(plotcolor);
                    _rectangle(_GFILLINTERIOR,20,5,480,290);
                    break;
 case 'A': case 'a': plotmode='a';
                    break;
 case 'T': case 't':  trig=1;
```

```
                break;
case 'C': case 'c': trig=0;
                break;
case 0:
   chin=getch();
   switch(chin)
   {
      case RIGHT: gainupdown=1;
                gainplot();
                break;
      case LEFT: gainupdown=-1;
                gainplot();
                break;
      case UP: positionsign=1;
                positionplot();
                break;
      case DOWN: positionsign=-1;
                positionplot();
                break;
      case F1: if(zoom>1) zoom=zoom-1;
                _settextposition(1,60);
                printf("%i ",zoom);
                break;
      case F2: if(zoom<10) zoom=zoom+1;
                _settextposition(1,60);
                printf("%i ",zoom);
                break;
      case F4:       savedat();        /* Save step responses if need
                break;
      case F5:       yawangle=yawangle-yawanglestep;
                newmixer();
                break;
      case F6:       yawangle=yawangle+yawanglestep;
                newmixer();
                break;
      case F9: rota=0;
                _settextcolor(8);
                _settextposition(19,75);
                _outtext("on");
                _settextcolor(4);
                _settextposition(18,75);
                _outtext("off");

                break;
      case F10: rota=1;
                _settextcolor(8);
                _settextposition(18,75);
                _outtext("off");
                _settextcolor(4);
                _settextposition(19,75);
                _outtext("on");
                break;
}
break;
case 'E':case 'e':  chd=1;
                break;
case 'D':case 'd': chd=0;
                break;
case ESC: ch=ESC; break;
}
```

```
        }
        chin='.';
}
/*************************************************************************/
   initialize: setting the parameters of the dual phase advance compensator */
   void initialize(void)
{
   double dt,T,n,Tnd;
   int breakfreq;
   printf("\n Enter Compensator's Break frequency (170hz): ");
   scanf("%d",&breakfreq);          /* Get the break frequency */
   dt=1/((double) samplrate);       /* period of the sampling */
   T=1/(2.0*pi*(double)(breakfreq));        /* 1/ break frequency       */
   printf("\n \n T= %9.6f\n",T);
   n=10.0;                          /* ratio of lead to lag break frequency */

   a11=1/((dt*dt)+4*T*dt+4*(T*T));
   b=(dt*dt)+4*n*dt*T+4*n*n*T*T;
   c=2*dt*dt-8*n*n*T*T;
   d=(dt*dt)-4*n*dt*T+4*n*n*T*T;
   e=2*dt*dt-8*T*T;
   f=(dt*dt)-4*dt*T+4*T*T;
   printf("\n");
   printf("\n");
   printf("\n");
   printf("The compensator parameters are:\n");
   printf("a=%9.6f\n",a11);
   printf("b=%9.6f\n",b);
   printf("c=%9.6f\n",c);
   printf("d=%9.6f\n",d);
   printf("e=%9.6f\n",e);
   printf("f=%9.6f\n",f);

  getch();

 }

/*************************************************************************/
/* Plotdat: plot of signals */
void plotdat(void)
{
                kv=4+(j+1)*48-(int)(v*2*zoom);
               if(kv>=290) kv=290;
               _setcolor(plotcolor);
               _setpixel(ix,olddat[ix][j+1]);
               _setcolor(datcolor);
               _setpixel(ix,kv);
               olddat[ix][j+1]=kv;
 }
/*************************************************************************/
/* trigplot: plot of signals */
void trigplot(void)
{    int plotcenter=148;

                kv=plotcenter-(int)(v*10*zoom);
               if(kv>=290) kv=290;
               _setcolor(plotcolor);
               _setpixel(ix,olddat[ix][j+1]);
               _setcolor(datcolor);
               _setpixel(ix,kv);
```

```
                olddat[ix][j+1]=kv;
/*              tmp[j]=kv;                      */
 }
/*****************************************************************/
   gainplot: to display gain on each degree of freedom graphically */
void gainplot(void)
{
        int gainknob,knobcolor;
        switch(ch2)
        {
           case 'x':
              if((gain[0]>=0) && (gain[0]<=2))
               {
                knobcolor=3*(gainupdown+1);
                _setcolor(knobcolor);
                gainknob=40+(int)(gain[0]*80.0);
                _rectangle(_GFILLINTERIOR,gainknob,320,gainknob+3,340);
                gain[0]=gain[0]+gainupdown*gainstep;
                _settextposition(21,27);
                printf("%4.3f",gain[0]);

               }
              else
                  if(gain[0]<0) gain[0]=0;
                  if(gain[0]>2) gain[0]=2;
               break;
           case 'y':
               if((gain[1]>=0) && (gain[1]<=2))
                {
                _setcolor(knobcolor);
                gainknob=40+(int)(gain[1]*80.0);
                _rectangle(_GFILLINTERIOR,gainknob,352,gainknob+3,372);
                gain[1]=gain[1]+gainupdown*gainstep;
                _settextposition(23,27);
                printf("%4.3f",gain[1]);
               }
              else
                  if(gain[1]<0) gain[1]=0;
                  if(gain[1]>2) gain[1]=2;
            break;
           case 'z':
               if((gain[2]>=0) && (gain[2]<=2))
                {
                knobcolor=3*(gainupdown+1);
                _setcolor(knobcolor);
                 gainknob=40+(int)(gain[2]*80.0);
                 _rectangle(_GFILLINTERIOR,gainknob,384,gainknob+3,404);
                 gain[2]=gain[2]+gainupdown*gainstep;
                 _settextposition(25,27);
                 printf("%4.3f",gain[2]);
                }
              else
                  if(gain[2]<0) gain[2]=0;
                  if(gain[2]>2) gain[2]=2;
            break;
           case 'm':
              if((gain[3]>=0) && (gain[3]<=2))
              {
                knobcolor=3*(gainupdown+1);
                _setcolor(knobcolor);
```

```
            gainknob=40+(int)(gain[3]*80.0);
            _rectangle(_GFILLINTERIOR,gainknob,416,gainknob+3,436);
            gain[3]=gain[3]+gainupdown*gainstep;
            _settextposition(27,27);
            printf("%4.3f",gain[3]);
          }
          else
              if(gain[3]<0) gain[3]=0;
              if(gain[3]>2) gain[3]=2;
        break;
       case 'n':
                  if((gain[4]>=0) && (gain[4]<=2))
          {
            knobcolor=3*(gainupdown+1);
            _setcolor(knobcolor);
            gainknob=40+(int)(gain[4]*80.0);
            _rectangle(_GFILLINTERIOR,gainknob,448,gainknob+3,468);
            gain[4]=gain[4]+gainupdown*gainstep;
            _settextposition(29,27);
            printf("%4.3f",gain[4]);
          }
           else
              if(gain[4]<0) gain[4]=0;
              if(gain[4]>2) gain[4]=2;
                            break;


    }
}
/*****************************************************************/
/* positionplot: control and display of position knobs           */
void positionplot(void)
{
    int knobcolor,positionknob,bkcolor;
    knobcolor=10;
    bkcolor=4;
     if(trig==1)
        ix=25;
        sampler=0;  /* counter for collecting data in ISR */

     switch(ch2)
     {

        case 'x':
             _setcolor(0);
             _rectangle(_GFILLINTERIOR,310,320,461,340);
             _setcolor(bkcolor);
             _rectangle(_GBORDER,310,320,461,340);
             _setcolor(knobcolor);
             position[0]=position[0]+positionsign*positionstep;
             if(position[0]<-1) position[0]=-1;
             if(position[0]>1) position[0]=1;
             positionknob=385+(int)(position[0]*75.0);
             _rectangle(_GFILLINTERIOR,positionknob,320,positionknob+1,340);
             break;
        case 'y':
             _setcolor(0);
             _rectangle(_GFILLINTERIOR,310,352,461,372);
             _setcolor(bkcolor);
```

```
            _rectangle(_GBORDER,310,352,461,372);
            _setcolor(knobcolor);
            position[1]=position[1]+positionsign*positionstep;
            if(position[1]<-1) position[1]=-1;
            if(position[1]>1) position[1]=1;
            positionknob=385+(int)(position[1]*75.0);
            _rectangle(_GFILLINTERIOR,positionknob,352,positionknob+1,372);
            break;
        case 'z':
            _setcolor(0);
            _rectangle(_GFILLINTERIOR,310,384,461,404);
            _setcolor(bkcolor);
            _rectangle(_GBORDER,310,384,461,404);
            _setcolor(knobcolor);
            position[2]=position[2]+positionsign*positionstep;
            if(position[2]<-1) position[2]=-1;
            if(position[2]>1) position[2]=1;
            positionknob=385+(int)(position[2]*75.0);
            _rectangle(_GFILLINTERIOR,positionknob,384,positionknob+1,404);
         break;
        case 'm':
            _setcolor(0);
            _rectangle(_GFILLINTERIOR,310,416,461,436);
            _setcolor(bkcolor);
            _rectangle(_GBORDER,310,416,461,436);
            _setcolor(knobcolor);
            position[3]=position[3]+positionsign*positionstep;
            if(position[3]<-1) position[3]=-1;
            if(position[3]>1) position[3]=1;
            positionknob=385+(int)(position[3]*75.0);
            _rectangle(_GFILLINTERIOR,positionknob,416,positionknob+1,436);
         break;
        case 'n':
            _setcolor(0);
            _rectangle(_GFILLINTERIOR,310,448,461,468);
            _setcolor(bkcolor);
            _rectangle(_GBORDER,310,448,461,468);
            _setcolor(knobcolor);
            position[4]=position[4]+positionsign*positionstep;
            if(position[4]<-1) position[4]=-1;
            if(position[4]>1) position[4]=1;
            positionknob=385+(int)(position[4]*75.0);
            _rectangle(_GFILLINTERIOR,positionknob,448,positionknob+1,468);
            break;


    }
}
/*****************************************************************/
/* timersetup: sets up the sample rate clock on the second ADC for interrupt */
void timersetup(void)
{
   int rate;
/*    Timer ic AM9513 setting     */


        outp(ADC_2+1,0xdf); /* disarms counters */

        outp(ADC_2+1,23);   /* sets next outp to the
```

may needs

```
                                    master register*/

          outp(ADC_2,0xc0);    /* master rgister low byte */
          outp(ADC_2,0x41);    /* master register high byte */

          outp(ADC_2+1,0x01); /* sets next outp to
                                 the counter 1 register */

          outp(ADC_2,0x22);    /* sets wave form and base frequency */
          outp(ADC_2,0x0c);

            rate=(int)(31250/samplrate);

          outp(ADC_2+1,0x09); /* sets next outp to the load register */
          outp(ADC_2,rate);    /* it will count down from rate
                                 and set off an interrupt */

          outp(ADC_2+1,0x61); /* load and arm  */


}

/*****************************************************************/
/* savedat() saves data to file    */
void savedat(void)
{
      char fname[12];
       int i,j;
       FILE *fp;
    _settextposition(25,64);
    printf("Save data? y/n\n");
    ch=getch();
    if((ch=='Y') || (ch=='y'))
       {
         _settextposition(25,64);
         printf("Enter File Name");
         _settextposition(26,64);
         scanf("%s",&fname);
         if((fp=fopen(fname,"w+"))!=NULL)
           {
             fprintf(fp,"stepdat=[\n");
             for(i=1;i<=200;i++)
                   {
                     for(j=0;j<=4;j++)
                     fprintf(fp,"%le ",savdat[j][i]);
                     fprintf(fp,"\n");
                   }
             fprintf(fp,"];\n");
             fprintf(fp,"gains=[\n");
             for(i=0;i<=4;i++)
             fprintf(fp,"%le \n",gain[i]);  /* Save the gains */
             fprintf(fp,"];\n");
             fclose(fp);
           }
          else
          printf("Oops file error");
       }
        _settextposition(25,64);
        printf("                ");
        _settextposition(26,64);
```

```
        printf("                ");

 }


/*************************************************************************/
/* cosmak : calculates cos values for angles 0..360                      */
void cosmak(void)
{     int i,number;
      double radang;

     maxnum=(int)(360.0/yawanglestep);
     yawangle=0;
     for(i=0;i<=maxnum;i++)
        {
          radang=(yawangle/180.0)*pi;
          cosdat[i]=cos(radang);
          yawangle=yawangle+yawanglestep;
        }
   yawangle=0;
}
/*************************************************************************/
/* newmixer : calculates new mixing matrix and current distribution */
void newmixer(void)
{  int i,j,k,nextone,index1,index2,index3,index4,index5,seventy2;
   int angindex;
   double angstep=6, yawangletemp;
   double mixtemp[5][5],interpol;

   angindex=(int)(yawangle/yawanglestep);       /* Index of the yaw angle  */
   seventy2=(int)(72.0/yawanglestep);           /* Index of 72 deg based on*/
   nextone=1;                                          /* the yaw step chosen
   if (angindex<0)
     {
       angindex=maxnum+angindex;
       nextone=-1;
     }
   index1=angindex;                    /*  determination of cosine matrix    */
   index2=abs(seventy2-angindex);      /*  index for calculating the zero
   index3=abs(2*seventy2-angindex);    /*  position current.  Every index*
   index4=abs(3*seventy2-angindex);    /*  is 72 degrees apart from its
   index5=abs(4*seventy2-angindex);    /*  adjasent neighbor.

   if(index1>maxnum)                   /* The five if statements rap around */
      index1=index1-maxnum;
   if(index2>=maxnum)                  /* the indceis to the begining after */
      index2=index2-maxnum;
   if(index3>=maxnum)                  /* 360 degrees rotation  */
      index3=index3-maxnum;
   if(index4>=maxnum)
      index4=index4-maxnum;
   if(index5>=maxnum)
      index5=index5-maxnum;


   offset[0]=initialoffset[0]*cosdat[index1];  /*  Levitation current calculatio
   offset[1]=initialoffset[0]*cosdat[index2];  /*  using the cosine array, which
   offset[2]=initialoffset[0]*cosdat[index3];  /*  selected depending on the ang
   offset[3]=initialoffset[0]*cosdat[index4];  /*  of the model.  Initialoffset[
   offset[4]=initialoffset[0]*cosdat[index5];  /*  is the highest steady current
```

```
   yawangletemp=fabs(yawangle);

   if(yawangle>=360)                          /* Adjustment for angles>360  */
       yawangletemp=yawangle-360;

   ang=(int)(yawangletemp/angstep);      /* Determination of number of steps */
   nextone=1;
/* Determination of the interpolating coefficent */
   interpol=((yawangletemp)-(double)((ang)*angstep))/angstep;

    if(yawangle<0)
    {
      nextone=-1;
      ang=60-ang;
    }

/* Interpolation between the mixing matrices */
     for(i=0;i<=4;i++)
         {
             for(j=0;j<=4;j++)
                 {
                        mix[i][j]=interpol*(mixall[i][j][ang+nextone]
                        -mixall[i][j][ang])
                        +mixall[i][j][ang];
                 }
         }
        _settextposition(22,72);
       printf("%5.2f",yawangle);



/*************************************************************************/
/* Intrrupt service routine  */
/* datin() obtains data from ADC in the array                    */
void _interrupt _far datin(void)
{
    unsigned int y11,Outport;
    int k,df;
   for(k=0;k<=4;k++)
        {
              mode = 7;
              flag = 0;
              params[0] = k;
              params[1] = 0;
             mscl_das40 (&mode, params, &flag);
              if (flag != 0) process_error();
              y=params[0];
              vsensor[k]=(float)(y/409.5);
        }
_enable()       ;

/*************************************************************************/
/*    decoupler                                  */
    v1[2]=vsensor[1]-5;
    v1[5]=((vsensor[3]+vsensor[4]-10)*-0.5);
    v1[8]=((vsensor[0]+vsensor[2]-10)*0.5);
    v1[11]=(vsensor[0]-vsensor[2]);
    v1[14]=(vsensor[3]-vsensor[4]);
```

Single A/D conversion.
channel
error
← LP_ADC_VALUE
Volts
may need adjusting if just using 1 MBA

```
    if(sampler<=200)
    {
        for(df=0;df<=4;df++)
        savdat[df][sampler]=v1[2+df*3];
        sampler++;
    }
 /* compensator: dual phase advance using backward difference method */
 /*   v1[2+3*l]=v1[2+3*l]+position[l]*4;   */

      for(l=0;l<=4;l++)
                {
                 v2[2+3*l]=a11*(gain[l]*(b*(v1[2+3*l]+position[l])
            +c*v1[1+3*l]+d*v1[3*l])-
                          e*v2[1+3*l]-f*v2[3*l]);
                }
/*********************************************************************/
/* current formation    */
    for(l=0;l<=4;l++)
       {
          current[l]=v2[2]*mix[l][0]+v2[5]*mix[l][1]+v2[8]*mix[l][2]+
                     v2[11]*mix[l][3]+v2[14]*mix[l][4]+offset[l];
/*********************************************************************/
/* Data outputs to the five D/As    */
        if(fabs(current[l])<10.0)
          y11=(unsigned int)(204.75*(current[l]+10.0));
         Outport=DDAbase+l*2;
_disable();
          _asm
               {
                  push ax;
                  push dx;
                  mov dx,Outport;
                  mov ax,y11;
                  out dx,ax;
                  pop dx;
                  pop ax;
               }
       _enable();
           v1[3*l]=v1[1+3*l];
           v1[1+3*l]=v1[2+3*l];
           v2[3*l]=v2[1+3*l];
           v2[1+3*l]=v2[2+3*l];
       }
   _disable();

   _asm
       {
            push ax
            push dx

            mov dx,ADC_2      ;  Load the ADC board address
            add dx,6          ;  point to adc high byte register
            or al,al          ; clear al
            in al,dx          ;  output command

            mov dx,20h          ; load 8259 address
            mov al,20h          ; set normal priority, EOI=1, (65h)
            out dx,al           ; output command
```

Volts.

channel / board address

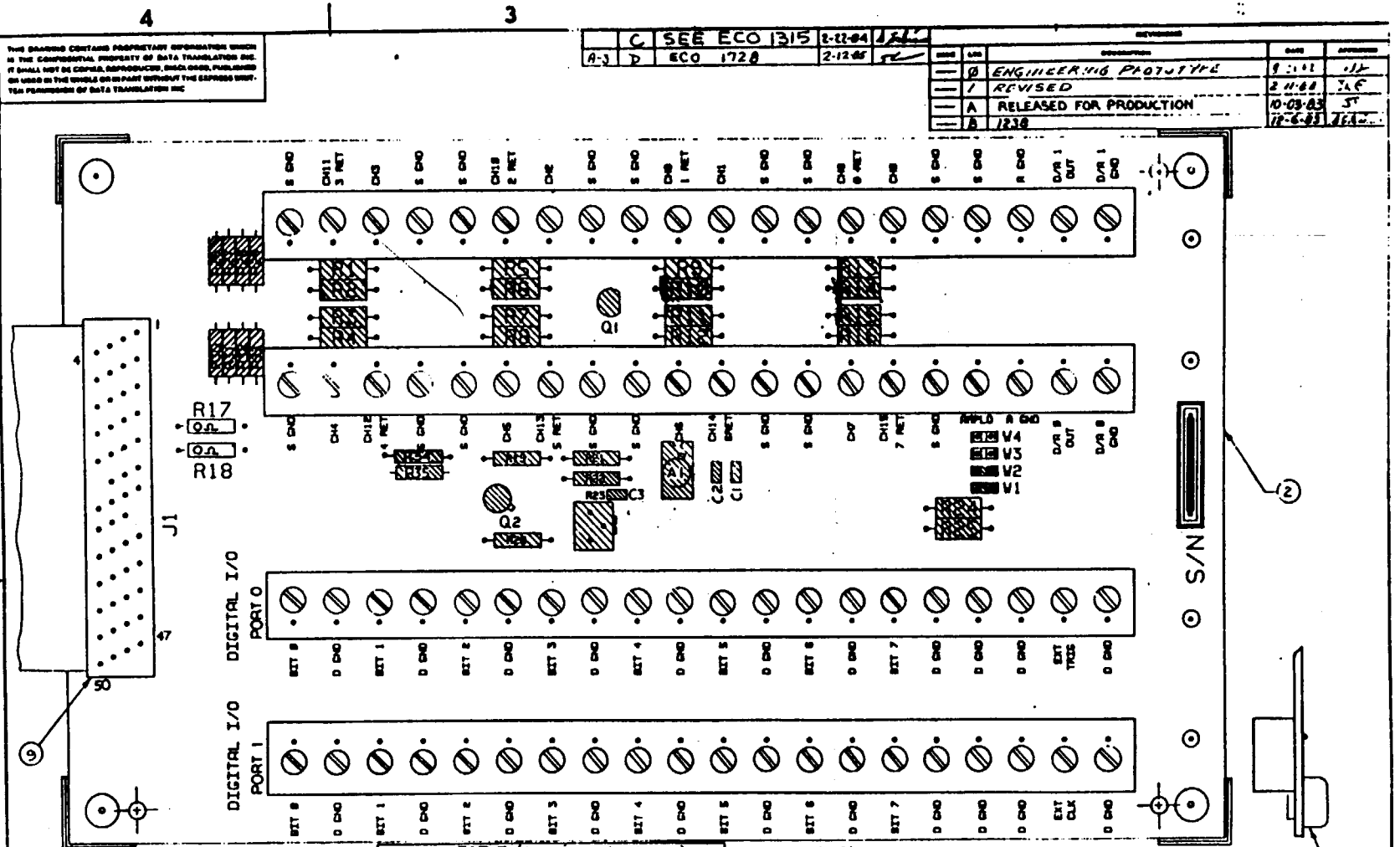
LP_ DAC_ VALUE

Timer.

```
            pop dx
            pop ax
        }
    _enable();
}
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*************************************************************************/
void adcinitialize(void)
{
    /* DAS-40 Initialization  */
    system("d40");
    mode=0;
    flag=0;
    params[0]=0;
    mscl_das40(&mode,params,&flag);
    if(flag !=0) process_error();
    _clearscreen(_GCLEARSCREEN);

}
/*************************************************************************/
void process_error()
{
    putch(7); putch(7);
    printf ("**** Error %u detected in mode %u      ", flag & 0xff, ((flag & 0xf
    exit(1);
}
/*************************************************************************/
void acknowledge(void)

    _asm
        {
            push ax
            push dx

            mov dx,ADC_2      ;  Load the ADC board address
            add dx,6          ;  point to adc high byte register
            or al,al          ; clear al
            in al,dx          ;  output command


;           mov dx,0a0h          load second 8259 address
;           mov al,20h           set normal priority, EOI=1, (65h)
;           out dx,al            output command

            mov dx,20h          ; load 8259 address
            mov al,20h          ; set normal priority, EOI=1, (65h)
            out dx,al           ; output command


            pop dx
            pop ax
        }
}

/*************************************************************************/
void shutdown(void)
 {
          int dtoa;
          _dos_setvect(intnum,oldnum);  /* reset interrupt vector table */
```

```
            for(dtoa=0;dtoa<=4;++dtoa)
                {
                outp(DDAbase+dtoa*2,0);    /*  0000000 for low byte at zero  */
                outp(DDAbase+dtoa*2+1,8); /* 1000 for high byteat zero */
                }
            outp(ADC_2+1,0xdf);  /* shut down interrupt counter  */
            system("d40 u");

}
/*****************************************************************/
/* getdat()  loads mixing matrix data   */

void getdat(void)
{
     char fname[12];
     int i,j,k,dim;
     FILE *fp;
     _settextposition(14,25);
     printf("Enter Data File Name");
     _settextposition(15,30);
     scanf("%s",&fname);
     if((fp=fopen(fname,"r+"))!=NULL)
     {
         printf("Please wait for data to load\n");
         fscanf(fp,"%d",&dim);
         printf(" Number of matrices are: %d \n ",dim);
         for(k=0;k<=dim-1;k++)
         {
            for(i=0;i<=4;i++)
            {
                for(j=0;j<=4;j++)
                    fscanf(fp,"%lf",&mixall[i][j][k]);

             }
         }
        printf(" Data was successfully loaded, hit Enter");
        getch();
         fclose(fp);
     }
          else
          fprintf(stderr,"Oops file error");

}
```

Figure 4



	C	SEE ECO 1315	2-22-84
A-3	D	ECO 1728	2-12-85

—	Ø	ENGINEERING PROTOTYPE	9-1-82	
—	1	REVISED	2-4-83	
—	A	RELEASED FOR PRODUCTION	10-03-83	
—	B	1238	12-6-83	

707	
REF DES	DESCRIPTION
—	BARRIER STRIP
J1	CABLE ASSY
R17, R18	0 Ω
W3, W4	JUMPER & PINS

707-M	
—	BARRIER STRIP
J1	CABLE ASSY
R17, R18	0 Ω
W3, W4	JUMPER & PINS
R1, R4, R5, R8, R9, R12, R13, R16	250 Ω .05%
R2, R3, R6, R7, R10 R11, R14, R15	10K

707-T (PLUS THE FOLLOWING)	
REF DES	DESCRIPTION
A1	OP21GP
C1, C2, C3	.1UF
R2, R3, R6, R7, R10, R11, R14, R15, R21	10K
R19	169K
R20	10.7K
R22	TRIM
R23	500 Ω
R24	4K .05%
R25	1K .02%
R26 THRU R33	100MEG
R35	12K
Q1	2N5088
Q2	LH0070-0H
W1, W2	JUMPER & PINS
R34	2.2K

NOTES:
1. R22 TO BE MATCHED TO Q1
2. FOR LIST OF MATERIALS SEE A SIZE DWG 00818

DATA TRANSLATION
MARLBORO, MASSACHUSETTS 01752

TITLE: COMPONENT BOARD ASSEMBLY DT707 (TABULATED)

N/A	N/A	C	00817 D
00818	DT707	SCALE NONE	SHEET 1 OF 1

From Bohr and Hyat's stored magnetic energy theory, the force of attraction for one electromagnet is,

$$F := \frac{1}{2 \cdot \mu_o} \cdot B^2 \cdot A_T = (1/3)\ m\ g$$

$$F := \frac{A_T}{2 \cdot \mu_o} \cdot \left[\frac{\mu_o \cdot N \cdot (i + I)}{2 \cdot [g_o + y]} \right]^2$$

$$F := \frac{A_T}{8 \cdot \mu_o} \cdot \left[\frac{\mu_o \cdot N \cdot i}{[g_o + y]} + \frac{\mu_o \cdot N \cdot I_o}{[g_o + y]} \right]^2$$

$$g := g_o + y$$

Expanding the squared term

$$F := \frac{A_T}{8 \cdot \mu_o} \cdot \left[\frac{[\mu_o \cdot N \cdot i]^2}{g^2} + \frac{2 \cdot \mu_o^2 \cdot N^2 \cdot I_o \cdot i_o}{g^2} + \frac{[\mu_o \cdot N \cdot I_o]^2}{g^2} \right]$$

Collecting similiar terms

$$F := \frac{A_T \cdot \mu_o \cdot N^2}{8 \cdot g_o^2} \cdot \left[i^2 + 2 \cdot i \cdot I_o + I_o^2 \right]$$

Since, $g := g_o + y$ then, $g^2 := g_o^2 + 2 \cdot g_o \cdot y + y^2$

$$F := \frac{A_T \cdot \mu_o \cdot N^2 \cdot \left[i^2 + 2 \cdot i \cdot I_o + I_o^2 \right]}{8 \cdot \left[y^2 + 2 \cdot g_o \cdot y + g_o^2 \right]}$$

$A_T := A_1 + A_2$ ---> $A_1 := A_2 = A$ ---> $A_T := 2 \cdot A$

Resulting F is the total force acting on the rim,

$$F := \frac{A \cdot \mu_o \cdot N^2 \cdot \left[i^2 + 2 \cdot i \cdot I_o + I_o^2 \right]}{4} \cdot \left[\frac{1}{y^2 + 2 \cdot g_o \cdot y + g_o^2} \right] = (c1/4)(c2)$$

From Taylor's Series Expansion at the equilibrium point (y ,i),

$$F(y,i) := F\begin{bmatrix} y & , i \\ o & o \end{bmatrix} + \begin{bmatrix} y - y & , i - i \\ & o \qquad o \end{bmatrix} \cdot FGradient\begin{bmatrix} y & , i \\ o & o \end{bmatrix}$$

$$F(y,i) := y \cdot \frac{\delta F}{\delta y} \cdot \begin{bmatrix} y & , i \\ o & o \end{bmatrix} + i \cdot \frac{\delta F}{\delta i} \cdot \begin{bmatrix} y & , i \\ o & o \end{bmatrix}$$

$$\frac{\delta F}{\delta y} := \left[\frac{\delta}{\delta y} \cdot \left[\frac{A}{4}\right]\right] \cdot (c2) + \left[\frac{\delta}{\delta y} \cdot (c2)\right] \cdot \left[\frac{c1}{4}\right]$$

$$\frac{\delta F}{\delta y} := \frac{-2 \cdot \left[y + g_o\right]}{y^2 + 2 \cdot y \cdot g_o + g_o^2} \cdot \left[\frac{c1}{4}\right]$$

$$\left[y^2 + 2 \cdot y \cdot g_o + y^2\right] := y^4 + 4 \cdot y^3 \cdot g_o + 6 \cdot y^2 \cdot g_o^2 + 4 \cdot y \cdot g_o^3 + g_o^4$$

$$\frac{\delta F}{\delta y} := \left[\frac{-\left[y + g_o\right] \cdot c1}{2 \cdot \left[y^4 + 4 \cdot y^3 \cdot g_o + 6 \cdot y^2 \cdot g_o^2 + 4 \cdot y \cdot g_o^3 + g_o^4\right]}\right]$$

At equilibrium, $i := 0$ & $y := 0$

$$\frac{\delta F}{\delta y} := \frac{-A \cdot \mu_o \cdot N^2 \cdot I_o^2}{2 \cdot g_o^3}$$

So then,

$$\frac{\delta F}{\delta i} := \frac{1}{4} \cdot \left[\frac{\delta}{\delta i} \cdot (c1)\right] \cdot c2 + \left[\frac{\delta}{\delta i} \cdot (c2)\right] \cdot c1$$

$$\frac{\delta F}{\delta i} := \frac{1}{4} \cdot \frac{A \cdot \mu_o \cdot N^2}{y^2 + 2 \cdot y \cdot g_o + g_o^2} \cdot 2 \cdot \left[i + I_o\right]$$

At equilibrium, $i := 0$ & $y := 0$

$$\frac{\delta F}{\delta i} := \frac{A \cdot \mu_o \cdot N^2 \cdot I_o}{2 \cdot g_o^2}$$

$$F(y,i) := \frac{-A \cdot \mu_o \cdot N^2 \cdot I_o^2}{2 \cdot g_o^3} \cdot y + \frac{A \cdot \mu_o \cdot N^2 \cdot I_o}{2 \cdot g_o^2} \cdot i$$

For the V-I relationship of the actuator,

first consider Kirchoff's voltage law for the bearing system,

$$V := R \cdot i + \frac{d}{dt} \cdot (L \cdot i)$$

$$V := R \cdot i + \left[L \cdot \frac{di}{dt} \right] + i \cdot \frac{dl}{dy} \cdot \frac{dy}{dt}$$

then by seperating the volatage, current and gap distance into the biased components and contolled components,

$$V := V_b + v \qquad i := I_b + i \qquad y := g_o + y$$

°So then,

$$V_b + v := R\cdot\left[I_b + i\right] + L\cdot\frac{di}{dt} + i\cdot\left[\frac{dl}{dy}\cdot\frac{dy}{dt}\right] + I_b\cdot\left[\frac{dl}{dy}\cdot\frac{dy}{dt}\right]$$

When $y := g_o$

then $V_b := R\cdot I_b$ assuming $i \equiv 0 \;\&\; y \equiv 0$

therefore,

$$i\cdot\left[\frac{dL}{dy}\cdot\frac{dy}{dt}\right] := 0$$

And, since

$$\frac{dL}{dy} := \frac{2\cdot N^2\cdot A\cdot \mu_o\cdot g_o}{g_o^2 - y^2}\cdot y$$

$$V(t) := R\cdot i(t) + L\cdot\frac{di(t)}{dt}$$

By taking the Laplace transform, $V(s) := R\cdot I(s) + L\cdot s\cdot I(s)$

when rearanging terms, $I(s) := \frac{V(s)}{R + L\cdot s}$

So then, from Newton's 2nd Law,

$$m \cdot \frac{d^2 \cdot y}{dt^2} := \frac{A \cdot \mu_o \cdot N^2 \cdot I_o^2}{2 \cdot g_o^2} \cdot i + \frac{-A \cdot \mu_o \cdot N^2 \cdot I_o^2}{2 \cdot g_o^3} \cdot y \qquad = c3\ i + c4\ y$$

$$m \cdot s^2 \cdot Y(s) := B \cdot Y(s) + A \cdot I(s)$$

recall, $\quad I(s) := \dfrac{V(s)}{R + L \cdot s}$

$$Y(s) \cdot \left[m \cdot s^2 + \frac{A \cdot \mu_o \cdot N^2 \cdot I_o^2}{2 \cdot g_o^3} \right] := V(s) \cdot \left[\frac{1}{L \cdot s + R} \right] \cdot \frac{A \cdot \mu_o \cdot N^2 \cdot I_o}{2 \cdot g_o^2}$$

$$\frac{Y(s)}{V(s)} := \frac{\dfrac{A\cdot\mu_o\cdot N^2\cdot I_o}{2\cdot g_o^2\cdot(L\cdot s+R)}}{m\cdot s^2+\dfrac{A\cdot\mu_o\cdot N^2\cdot I^2}{2\cdot g_o^3}}$$

dividing the 'm' out

$$\frac{Y(s)}{V(s)} := \frac{\dfrac{A\cdot\mu_o\cdot N^2\cdot I_o}{2\cdot g_o^2\cdot m\cdot\left[s+\frac{R}{L}\right]}}{s^2+\dfrac{A\cdot\mu_o\cdot N^2\cdot I^2}{2\cdot g_o^3\cdot m}}$$

$$\frac{Y(s)}{V(s)} := \frac{\dfrac{A \cdot \mu_o \cdot N^2 \cdot I_o^2}{2 \cdot g_o \cdot m}}{\left[s + \left[\frac{R}{L}\right]\right] \cdot \left[s^2 + \dfrac{A \cdot \mu_o \cdot N^2 \cdot I_o^2}{2 \cdot g_o^3 \cdot m}\right]}$$

Therefore, $$\frac{Y(s)}{V(s)} := \frac{c5}{(s + c6) \cdot \left[s^2 + c7\right]}$$

The variables defined are:

$$A := \pi \cdot \left[1.8 \cdot \left[\frac{2.54}{1}\right] \cdot \left[\frac{1}{100}\right]\right]^2 \qquad A = 0.007$$

$$\mu_0 := 4 \cdot \pi \cdot 10^{-7} \qquad \mu_0 = 1.257 \cdot 10^{-6}$$

$$N := 1619$$

$$I_0 := 0.57$$

$$g_0 := 0.0075946$$

$$m := \frac{21.59099}{3} \qquad m = 7.197$$

$$L := \frac{N^2 \cdot A \cdot \mu_0}{2 \cdot g_0} \cdot \begin{bmatrix} 1 \\ - \\ 2 \end{bmatrix} \qquad L = 0.712$$

$$R := 8$$

Derivation of the group constants:

$$c5 := \frac{A \cdot \mu_0 \cdot N^2 \cdot I_0}{2 \cdot g_0 \cdot m^2} \qquad c5 = 14.851$$

$$c6 := \frac{R}{L} \qquad c6 = 11.235$$

$$c7 := \frac{A \cdot \mu_0 \cdot N^2 \cdot I_0^2}{2 \cdot g_0 \cdot m^3} \qquad c7 = 1.115 \cdot 10^3$$

Therefore the plant dynamics equation or the open loop transfer function is:

$$G(s)_p := \frac{14.851}{(s + 11.235) + \left[s^2 + 1115\right]}$$